TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto

Jukka-Pekka Juntunen

Konekäännösjärjestelmän mukautumiskyvyn parantaminen älykkään käännösmuistin avulla

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten.

Työn valvoja: professori Markku Syrjänen

Työn ohjaaja: Ph.D. Harri Arnola (Kielikone Oy)

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä ja työn nimi:

Jukka-Pekka Juntunen

Konekäännösjärjestelmän mukautumiskyvyn parantaminen älykkään käännösmuistin avulla

Päivämäärä: 03.02.1996 **Sivumäärä:** 66

Osasto:
Tietotekniikan osasto / Tik-laitos

Professuuri:
Tik-93 Tietämystekniikka

Työn valvoja: Professori Markku Syrjänen
Työn ohjaaja: Ph.D. Harri Arnola

Tämän työn tavoitteena oli tutkia Kielikone Oy:n sääntöpohjaisen suomi-englanti konekäännösjärjestelmän - TranSmartin - käytetävyyteen ja mukautumiskykyyn liittyviä ongelmia. Työssä selvitettiin näiden ongelmien päätyypit sekä jotain ratkaisumahdollisuuksia. Työn konkreettinen tulos on älykäs käännösmuisti.

Käännösjärjestelmien mukautumisongelmat näyttävät jakautuvan kahteen päätyyppiin: kielellisiin ja tietoteknisiin ongelmiin. Älykäs käännösmuisti pyrkii pääasiassa lievittämään TranSmartin kielellisen oppimiskyvyn puutteita.

Käännösmuisti on tietokanta, johon talletetaan virkkeitä ja niiden käännöksiä. Käännösmuistista voidaan myöhemmin hakea käännettävien virkkeiden tai niitä lähellä olevien virkkeiden käännöksiä ja siten säästyä uudelleenkääntämisen vaivalta. Älykäs käännösmuisti toimii virkkeiden depedenssijäsennyksien perusteella. Tämän ansiosta virkkeiden haku on kielellisesti kattavampaa ja tarkempaa kuin perinteisissä käännösmuisteissa. Tämän lisäksi älykäs käännösmuisti sisältää käännösmuokkaimen, joka mahdollistaa talletettujen virkkeiden käännösten paremman hyödyntämisen.

Puiden vertailu ja suurten tietomäärien hallitseminen nostaa esiin monia aikavaativuuteen liittyviä ongelmia käännösmuistin toteuksessa. Näitä on pyritty lieventämään tekemällä hausta monivaiheinen ja tallettamalla tietokantaan mahdollisimman hyvin hakuun soveltuvaa tietoa. Älykkään käännösmuistin tietokannan monitasoinen indeksointi perustuu jäsennyspuihin. Indeksoinnin avulla haku rajataan sopivaan joukkoon virkkeitä.

Indeksoinnin lisäksi tietokantaan talletettujen virkkeiden jäsennyspuut on normalisoitu siten, että niistä on poistettu syntaktisesti ja semanttisesti merkityksettömiä solmuja. Puiden solmujen järjestys on lisäksi sidottu solmujen relaatioiden mukaan. Tämän ansiosta varsinainen kahden jäsennyspuun etäisyyden laskeva algoritmi toimii käytännön virkkeillä lähes lineaarisessa ajassa suhteessa virkkeiden pituuteen.

Kun käännösmuistista on löydetty lähin esimerkkikäännös, käytetään tarvittaessa käännösmuokkainta käännöksen korjaamiseen. Käännösmuokkain paikantaa käännettävän ja esimerkkikäännöksen eroavat sanat ja niiden vastineet sekä korjaa kohdekielisen virkkeen käännettävää virkettä vastaavaksi.

Avainsanat: luonnollisen kielen tietotekniikka, konekääntäminen, käännösmuisti, esimerkkiperustainen konekääntäminen, tekstinhaku, käytettävyys

Author and name of the thesis:

Ikka-Pekka Juntunen

Intelligent Translation Memory to Improve the Adaption of a Machine Translation System

Date: 03.02.1996 **Number of pages:** 66

Department:
Department of Information Technology

Professorship:
Tik-93 Knowlegde Engineering

Supervisor: Professor Markku Syrjänen
Instructor: Ph.D. Harri Arnola

The objective of this thesis was to study the usability and adaption problems of the rule-based finnish-english machine translation system TranSmart developed in Kielikone Ltd. During the work, the main categories of the problems were discovered and some solutions were presented. The concrete outcome of the thesis is **Intelligent Translation Memory**.

All the adaption problems of the present machine translation systems seem to fall in one of the two main categories: linguistic problems and computer-related problems. The Intelligent Translation Memory tries to solve some of the problems related to linguistic adaption of the TranSmart machine translation system.

In general, translation memory is a database containing translation examples. It is utilized during translation work. Readymade target sentences are retrieved for source senteces or closely related source sentences to avoid retranslation. In the Intelligent Translation Memory, the retrieval of text segments is based on the linguistic dependency trees of the sentences. It makes the search more precise and gives a better recall rate than the traditional translation memories. In addition, the Intelligent Translation memory has an **Automatic Translation Editor** to correct the sentences returned.

Dependency tree comparison and management of large quantities of data make time resources the bottleneck of the Intelligent Translation Memory. Specialized datastructures are presented to make the search more efficient and to help comparison in many ways. The indexing of the database is multileveled and based on the features of the dependency trees. Indexing tables produce a set of candidates during the search.

In addition to the indexing, the dependency trees stored in the database are normalized. Some of the syntactically and semantically redundant nodes and attributes are removed. The ordering of the nodes in the tree is also fixed. This makes it possible for the actual tree distance calculation to work in almost linear time.

Once the best match is found in the database, the Automatic Translation Editor takes charge of the process. If the match is not a perfect one, the Editor locates the nonmatching nodes in the translation example. The target language sentence is fixed and the final translation is produced.

Keywords: natural language processing, machine translation, translation memory, example-based machine translation, full text seach, usability

SISÄLLYS

ALKUSANAT

Tämä diplomityö on tehty Kielikone Oy:ssä TranSmart-konekäännösprojektin yhteydessä. Työn valvojana toimi tietämystekniikan professori Markku Syrjänen ja ohjaajana Ph.D. Harri Arnola. Haluan kiittää heitä saamastani opastuksesta.

Työn valmistumiseen vaikuttivat monet henkilöt, erityisesti haluan kiittää Kielikoneen konekäännös-projektin henkilöstöä saamastani tuesta. Monien tässä työssä esitettyjen tulosten perusideat ovat Harri Arnolan kehittämiä.

Helsingissä, 3. helmikuuta 1996

Jukka-Pekka Juntunen

LYHENTEET JA MERKINNÄT

Kokekäännösjärjestelmiä ja -termejä

ATR	Advanced Telecommunications Reseach Japanissa.
Candide	IBM:n tilastolliseen menetelmään perustuva konekäännösjärjestelmä.
Eurolang Optimizer	Eurolangin käännösmuisti ja kääntäjän työasema.
FAHQT	Täysin automaattinen koneellinen kääntäminen (Fully Automated High Quality Translation).
HAMT	Ihmisavusteinen konekääntäminen (Human-Aided Machine Translation).
MAHT	Koneavusteinen ihmiskääntäminen (Machine-Aided Human Translation).
METAL	Siemensin konekäännösjärjestelmä.
TITUS	Institut Textile de Francessa vuonna 1970 kehitetty käännösjärjestelmä.
Transit	Starin käännösmuisti ja kääntäjän työasema.
TranSmart	Kielikone Oy:n sääntöperustainen suomi-englanti käännösjärjestelmä.
Translation Manager	IBM:n käännösmuisti ja kääntäjän työasema.
Translator's Workbench	Tradosin käännösmuisti ja kääntäjän työasema.
SINIX	Siemensin METAL-konekäännösjärjestelmän osa, joka tulkitsee asiakirjojen muotoiluja.
Verbmobil	Saksa-englanti käännösjärjestelmä puhutun kielen kääntämiseen.

Muut

AIX	IBM:n markkinoima ja käyttämä UNIX-käyttöjärjestelmä.
Ami Pro	MS Windows-käyttöjärjestelmän tekstinkäsittelyohjelma (valmistaja Lotus corporation).
Berkeley DB	Berkeleyn yliopistossa Kaliforniassa kehitetty tietokantaohjelmakirjasto.
DFD	Tietovuokaavio (Data Flow Diagram).
DX200	Erään Nokia Telecommunicationsin valmistaman puhelinkeskuksen tyyppi.
ER-kaavio	Tietokantojen kuvausformalismi (Entiry-Relation kaavio).
IBM PowerPC	IBM:n Unix-työasema.
IR	Tiedonhaku (Information Retrieval).
MS Word for Windows	MS Windows-käyttöjärjestelmän tekstinkäsittelyohjelma (valmistaja Microsoft corporation).
NTC	Nokia Telecommunications Oy.
OCR	Tekstin tunnistaminen optisella lukijalla paperiasiakirjoista luetusta kuvasta (Optical Character Recognition).
RTF	Laajalle levinnut Microsoft Corporationin asiakirjaformaatti (Rich Text Format).
SGML	ISO 8879-standardin kuvaama tiedon metakuvauskieli (Standard Generalized Markup Language).
simulated annealing	Minimin hakemiseen kehitetty heuristiikka, joka perustuu paikallisen minimin hakemiseen ja tarvittaessa sieltä ulospääsemiseen satunnaistekijän avulla.
WAIS	Internetissä käytetty tekstin hakuohjelma (Wide Area Information Server)..
WinMorfo	Kielikone Oy:n oikaisulukuohjelma.
Word Pro	MS Windows-käyttöjärjestelmän tekstinkäsittelyohjelma (valmistaja Lotus corporation).
WordPerfect	MS Windows-käyttöjärjestelmän tekstinkäsittelyohjelma (Novell corporation).
Xteelt	TranSmartin aputyökalu suomi-englanti -käännössanastojen kehittämiseen.

1. Johdanto

1.1. Työn aihe

Tämän työn pohjana ja testitympäristönä toimii **Kielikone Oy**:n sääntöpohjainen suomi-englanti konekäännösjärjestelmä **TranSmart**. TranSmart on melko tyypillinen käännösjärjestelmä: sen toiminta perustuu ennalta rajattuun malliin luonnollisesta kielestä, joka on sanastoltaan, rakenteeltaan ja muotoiluiltaan avoin kokonaisuus. Käännösprosessi onkin monivaiheinen ja ulkopuolisille usein vaikeaselkoinen. Kun tällainen tekninen järjestelmä otetaan kaupalliseen tuotantokäyttöön, nousee käytännössä helposti esiin poikkeustapauksia, joita alkuperäinen malli ei kata tai joita tuotteen kehitysaste ei vielä huomioi. Nämä käytännön ongelmat voidaan nimetä adaptiivisuusongelmiksi.

Jotkin konekäännösjärjestelmän adaptiivisuusongelmat juontavat juurensa luonnollisen kielen erikois- tai poikkeussäännöistä, jotka saattavat käytännössä olla harvinaisia. Toiset ongelmat ovat hyvin tavallisia ja esimerkiksi käytettyjen työkalujen, vaikkapa tekstinkäsittelyohjelmien tiedostomuotojen aiheuttamia. Kaikkien näiden ongelmien ratkaiseminen on tärkeää järjestelmän tulosteen laadun takaamiseksi käyttäjien silmissä. Tämän työn tavoitteena on parantaa TranSmartin **käytettävyyttä** ja sen soveltuvuutta asiakkaiden tarpeisiin.

1.2. Työn rakenne

Työn aluksi selvitetään hieman konekääntämisen taustaa ja historiaa. Sovellusalueen ja TranSmartin ominaisuuksien kuvaamisen jälkeen kolmannessa kappaleessa esitetään tarkemmin ne ongelmat, joihin käännösjärjestelmän tulisi mukautua. Kappaleessa selvitetään myös joitain valmiita ratkaisuja konekääntämisen adaptiivisuusongelmiin. Kappaleen lopussa kuvataan TranSmart-järjestelmän tarvitsemia muutoksia.

Esitetyistä ratkaisumahdollisuuksista valitaan toteutettavaksi **älykäs käännösmuisti**, jonka ominaisuudet kuvataan kappaleessa neljä. Seuraavassa kappaleessa esitetään käännösmuistin osajärjestelmien toteutus ja sen teoreettiset perusteet.

Käännösmuistin soveltuvuutta testataan kappaleessa kuusi. Lopuksi esitetään työn yhteenveto ja jatkokehittämismahdollisuuksia.

2. Konekääntämisen taustaa

2.1. Konekääntäminen kommunikaation välineenä

Konekäännösjärjestelmä vastaa ihmisten kommunikointitarpeeseen helpottamalla eri kieltä puhuvien ihmisten vuorovaikutusta. Vuorovaikutus on informaation eli viestien välittämistä [Booth, 1989]. Konekäännösjärjestelmää käytetään viestien kääntämiseen kieleltä toiselle.

Ideaalinen konekäännösjärjestelmä poistaisi erään nykyisen globaalisen kommunikointiyhteiskunnan ongelman - ihmisten välisen kielimuurin. Sille annettaisiin syötteenä viesti **lähdekielellä** ja tulosteena saataisiin viestin oikea käännös **kohdekielellä** nopeasti ja vaivattomasti. Ideaalinen käännösjärjestelmä tuottaisi nopeasti ja vaivattomasti käännöstuloksen, joka vastaisi viestien merkityksen tuntevan ihmiskääntäjän tekemää käännöstä.

2.2. Konekääntämisen lyhyt historia ja ongelmat

Ajatus kieltä kääntävästä koneesta on vanha, peräisin ainakin jo 1600-luvulta. [Hutchins, 1986].Varsinainen käännösjärjestelmien kehitys alkoi kuitenkin toden teolla vasta tietokoneiden keksimisen jälkeen kylmän sodan aikoihin 1950-luvulla, jolloin mm. sotateollisuuden tarpeet ja resurssit ajoivat järjestelmiä eteenpäin [Slocum, 1987].

Jo aivan ensimmäiset käännösjärjestelmien kehittäjät törmäsivät konekäännösjärjestelmien perimmäiseen ongelmaan: luonnollisen kielen rikkauteen ja sen alla piilevän semantiikan ongelmallisuuteen [Hutchins, 1986]. Kielellisten ilmaisujen määrä ja moninaisuus on valtava. Tämän lisäksi koneen, joka ei näe, kuule, tunne, haista, maista eikä varsinkaan omaa pelon, nälän, tyytyväisyyden tms. tunteita on hyvin vaikea **ymmärtää** kääntämäänsä tekstiä. Kuitenkin tekstin ymmärtäminen saattaa olla välttämätöntä oikean käännöksen löytymiselle. Kyseessä on siis tekoälyongelma, jolle ei löytyne suoraviivaista ratkaisualgoritmia.

Ongelmaa on kuitenkin yritetty ratkaista likimääräisesti monin eri keinoin. Ensimmäiset järjestelmät perustuivat vastinsanojen etsimiseen sanastosta ja käännöksen tekemiseen "sana-sanasta"-tyyliin, joka säilytti lähdekielen sanajärjestyksen. Tietyillä kielipareilla tekniikka saattoi joissain tapauksissa toimia, mutta vastaesimerkkejä on helppo keksiä. Tätä tekniikkaa laajennettiinkin sanaparien ja -joukkojen kääntämiseen, jolloin sanasto koostui esimerkiksi fraaseista, lauseita tai jopa lauseiden joukoista [Hutchins, 1986]. Tämä suoraviivainen tekniikka toimi teoriassa ideaalisilla kone- yms. resursseilla hyvin: sen avulla voitiin erotella eri tyyppiset ilmaisut varsin tehokkaasti, koska käännösyksikön kokoa voitiin tarpeen mukaan kasvattaa. Käyttämällä tarpeeksi pitkiä käännösyksiköitä onkin mahdollista rakentaa järjestelmä, joka ei tee paljon suoranaisia käännösvirheitä - se kääntää vain tuntemansa ilmaisut. Toisaalta käytännössä ei ole olemassa sellaista konetta, jonka muistiin mahtuisi tarpeeksi erilaisia ilmauksia [Nagao, 1986]. Tätä menetelmää voidaan kuitenkin käyttää täydentämään muita.

Suurten sanastojen ja säännöstöjen ongelmaa pyrittiin kiertämään **syventämällä** lähdekielisen tekstin lingvististä **analyysia**, minkä ansiosta voitiin käyttää yleisempiä sääntöjä, karsia vääriä tulkintoja ja saada kääntämistä helpottavaa lisätietoa käsiteltävästä aineistosta. Esimerkiksi Georgetown-IBM -projekti vuonna 1954 kuului ensimmäisiin, joissa analyysiä alettiin syventää [Hutchins, 1986].

Morfologisen analyysin ansiosta sanastoon talletettujen sanojen eri muotoja voitiin tehokkaasti karsia, koska käännettävän sanan perusmuoto voitiin etsiä sanan esiintymästä karsimalla sanojen morfeemit. Samoin kohdekielisen sanan muoto voitiin muodostaa

esimerkiksi lisäämällä prepositio tai taivuttamalla sanaa. Sanojen kaikkia muotoja ei siis tarvinnut enää erikseen luetella sanastossa.

Suurin puute suorassa sana-sanasta -kääntämisessä on sanojen välisten syntaktisten relaatioiden hyödyntämättä jääminen. Seurauksena ovat usein väärät sananvalinnat ja väärä sanajärjestys. Tästä syystä toteutettiin virkkeiden **syntaktinen analyysi**, jossa virkkeet jäsennettiin ennen kääntämistä kieliopin mukaisten sääntöjen perusteella esimerkiksi riippuvuuspuuksi. Samalla voitiin jo tehokkaasti karsia sanojen monitulkintaisuuksia.

Kieliopillista analyysiä voidaan vielä laajentaa **semanttiseksi analyysiksi**, jossa pyritään hakemaan esimerkiksi käännettävän virkkeen merkitystä esittävää graafia. Virkkeessä oleva tieto voidaan näin liittää taustatietoihin. Näin päästään jo lähelle ihmiskääntäjien toimintaa.

Sen sijaan että lisätään koneen älykkyyttä, voidaan myös rajoittaa ihmisten käyttämää kieltä. Esimerkiksi TITUS-järjestelmä toimii näin [Hutchins, 1986]. Järjestelmässä ei tarvita kuin rajattu sääntötietokanta ja sen toiminta on varmempaa. Tällaisten alikieliin perustuvien järjestelmien käyttöalue on kuitenkin rajoitettu.

2.3. Konekäännösjärjestelmien luokittelu

Nykyään konekääntämisessä voidaan erotella kaksi koulukuntaa: **transfer**- ja **interlinguajärjestelmät**. Transfer-järjestelmät ovat kieliparikohtaisia, kun taas interlinguajärjestelmät muuttavat käännettävät virkkeet ensin välikieleksi, interlinguaksi, josta sitten tuotetaan kohdekielinen viesti. Välikieli voi olla vaikkapa semantiikkaa esittävä graafi.

Järjestelmät voidaan jakaa myös niiden tietämyksen esitystavan mukaan sääntöperusteisiin ja korpusperusteisiin järjestelmiin. **Sääntöperusteisia järjestelmiä** opetetaan ihmisten laatimien käännössääntöjen avulla. **Korpusperusteiset** eli olemassaoleviin käännöksiin perustuvat järjestelmät etsivät kääntämiseen tarvittavan tietämyksen itse. Niiden avulla pyritään vähentämään asiantuntijaresursseja vaativaa kehitystyötä. Korpusperusteiset konekäännösjärjestelmät voidaan jakaa seuraaviin alaryhmiin:

1. **Tilastolliset** (Statistics Based) järjestelmät, jotka puhtaimmillaan perustuvat sanojen esiintymisen todennäköisyyteen.
2. **Esimerkkiperusteiset** järjestelmät (Example Based), joissa verrataan käännettävän tekstiyksikön ja talletettujen esimerkkikäännösten semanttista etäisyyttä ja sen avulla valitaan esimerkkikäännösten yhdistelmä.
3. **Neuraaliverkkoihin** (Connectionist) perustuvat järjestelmät. [Hutchins, 1993]

Näiden lisäksi voidaan tehdä myös muunlaisia jakoja mm. järjestelmän toteutustavan tai toimintaperiaatteiden mukaan.

2.4. Suomi-englanti käännösjärjestelmä TranSmart

Tämän työn lähtökohtana ja testausympäristönä toimii Kielikone Oy:n sääntöpohjainen suomi-englanti käännösjärjestelmä TranSmart. Järjestelmä perustuu muunnosmenetelmään. Niinpä sen toiminta jakautuu kolmeen perusvaiheeseen:

1. lähtökielen analyysiin,
2. muunnokseen lähtökieleltä kohdekielelle ja
3. kohdekielen synteesiin.

TranSmartin käännösprosessin ensimmäinen vaihe on virkkeen tunnistaminen ja virkkeen saneiden morfologinen analysointi. Analysoinnin aikana virkkeiden sanojen sananmuodot tulkitaan Morfo-ohjelman avulla. Näin saadusta sanojen morfologiset monitulkintaisuudet

esittävästä sanaketjusta muodostetaan depenssijäsennyspuu. Jäsennyspuussa sanojen monitulkintaisuudet on karsittu: jokaiselle saneelle on valittu sen oikea konteksista riippuva tulkinta sekä lauseenjäsenet ja sanojen väliset riippuvuudet on määritetty. Jäsennyksen jälkeen esimerkiksi tiedetään, että virkkeen *Voi on ruokaa* ensimmäinen sana on subjektina toimiva substantiivi *voi* eikä *voida*-verbi.

Oksaimen avulla jäsennyspuuta redusoidaan. Tuloksena saadaan yksinkertaisempi jäsennyspuu, josta virkkeen semanttinen rakenne käy selvemmin ilmi. Puun eräät puhtaasti syntaktiset solmut on karsittu tai muutettu toisten solmujen attribuuteiksi.

Analysointia seuraa muunnosvaihe. Muunnosvaiheen ensimmäinen askel, sanojen semanttinen leimaaminen, suoritetaan käännössanastojen avulla. Yhdyssanat paloitellaan tarvittaessa produktiivista kääntämistä varten. Ensin käännetään kiteytetyt termit ja niiden jälkeen erikois- ja yleissanastoista löytyvät sanat. Sanastomuunnosta seuraavat rakenne- ja attribuuttimuunnosvaiheet, joissa jäsennyspuita muokataan yleisten rakenteellisten ominaisuuksien mukaan.

Kohdekielisen virkkeen synteesivaiheen aluksi muutetaan eräät solmujen syntaktiset attribuutit lähdekielen sanoiksi tai sanojen taivutusmuodoiksi. Tämän jälkeen puu linearisoidaan ja tuotetaan kohdekielisen virkkeen pintamuoto.

Sisäisesti suurin osa käännösvaiheista on toteutettu virtuaalikoneen avulla. Jokaista vaihetta vastaa virtuaalikoneen säännöstö. Käännösprosessia ohjataan prosessitaulun avulla, jossa listataan suoritettavat käännösvaiheet.

TranSmart käännösjärjestelmä kääntää SGML-muodossa olevaa tekstiä. SGML-muotoilut kulkevat koko käännösvaiheen läpi sanojen attribuutteina, jotka periytyvät kohdekielisten sanojen attribuuteiksi. Näin tiedoston SGML-muotoilu säilyy [Kulikov, 1990 ja Jäppinen et al., 1993].

3. Konekäännösjärjestelmien mukautumiskyky

3.1. Konekääntämisen mukautumisongelmat

3.1.1. Ongelmien päätyypit

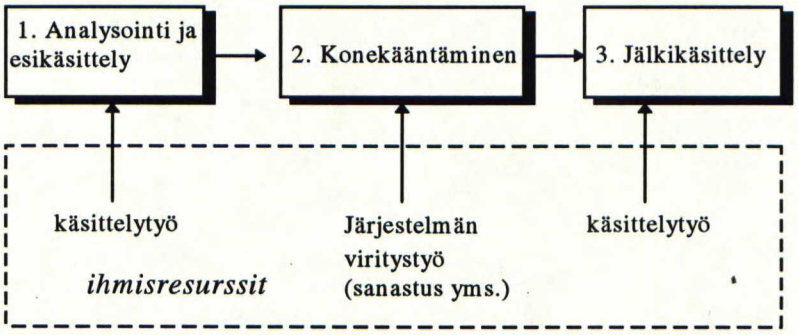
Mikään nykyisistä konekäännösjärjestelmistä ei ole ideaalinen, eikä edes lähellä sitä. Yksikään ei tuota aina ihmistasoista käännöstä vapaasti valitusta tekstistä. **Mukautumisongelmat** ovat konekääntämisen käytännön ongelmia, jotka johtuvat järjestelmien vaikeasti ymmärrettävistä toimintaperiaatteista ja jäykkyydestä sekä käännettävän aineiston suuresta kirjosta ja poikkeustapauksista. Osa näistä ongelmista voitaisiin ratkaista, jos käyttäjät voisivat itse parantaa järjestelmää ja sen toteutuksessa olisi otettu huomioon heidän tarpeensa. Tällainen järjestelmä olisi käytettävämpi. Osa adaptiivisuusongelmista voidaan ratkaista myös automaattisesti kehittämällä käännösjärjestelmää esimerkiksi tilastollisten menetelmien avulla.

Yleensä konekäännösjärjestelmät on suunniteltu kieliopillisesti oikein muodostettujen, virkkeistä koostuvien, tekstimuodossa olevien viestien kääntämiseen. Todellisuudessa ihmisten käyttämä **kieli** on usein epätäydellistä ja sisältää jopa virheitä. Käytetty kieli myös riippuu tilanteesta ja sovellusalueesta, kehittyy jatkuvasti sekä sisältää lukuisia poikkeussääntöjä. Sääntöpohjaiset käännösjärjestelmät vaativat lähtökieleltä usein virheetöntä kielioppia ja uusien käännössääntöjen lisääminen vaatii tietoa järjestelmän toiminnasta ja sen periaatteista. Järjestelmän tietämyksen lisääminen onkin usein mahdollista vain sen hyvin tunteville asiantuntijoille. Konekäännösjärjestelmän ytimeen on vaikeaa, turhauttavaa ja kallista rakentaa tietokantaa kaikista mahdollisista luonnollisen kielen ilmaisuista.

Käytetyn kielen lisäksi ihmisten toisilleen välittämien viestien **muoto** vaihtelee suuresti - ne sisältävät pelkkien virkkeiden lisäksi paljon muutakin tietoa. Viestit voivat koostua mm. kuvista, taulukoista, kaavoista, äänestä tai muista symboleista, jotka vaativat erityiskäsittelyä käännösjärjestelmässä. Viesti voi olla vaikkapa kokonainen sanomalehti palstoineen, kuvineen ja mainoksineen. Monet nykyiset konekäännösjärjestelmät joko vain kadottavat kuvat, muotoilut yms. oliot, tai niiden käännössäännöt eivät toimi lainkaan, jos viestissä on taulukoita tai kaavoja. Esimerkiksi TranSmart antaa seuraavan käännöksen: *Miten t ä m ä kääntyy?* → *How t ä ä m turns?*. Lisäksi viestin muodossa saattaa piillä kääntämistä helpottavaa tietoa, jota ei kannata hukata. Esimerkiksi otsikko on yleensä merkitty teksteissä omalla kirjasintyypillään, jonka avulla niiden tunnistaminen on helppoa. Näin voitaisiin otsikko tunnistaa helposti ja käännettäessä suomesta englantiin muuntaa sen kirjainten koot englannin kielen vaatimusten mukaisiksi.

3.1.2. Ongelmien seuraus - ihmistyö konekääntämisessä

Muunmuassa yllä kuvattujen ongelmien takia siihen koko prosessiin, joka vaaditaan lähdekielisen viestin kääntämiseen kohdekieliseksi, kuuluvat usein kuvassa 3.1 esitetyt ihmistyötä vaativat vaiheet [Nagao, 1989]. Työmäärä riippuu järjestelmästä, toisissa järjestelmissä kaikkia vaiheita ei tarvita:



Kuva 3.1. Konekäännösprosessin ihmistyötä vaativat vaiheet

1. Analysointi- ja esikäsittelyvaihe

Ensimmäisessä vaiheessa päätetään, kannattaako kyseessä olevan materiaalin käännöstyötä tehdä koneellisesti. Jos teksti katsotaan soveltuvaksi järjestelmälle, aloitetaan sen koneellinen kääntäminen.

Vaiheen aikana voidaan mm. profiloida tekstiä, etsiä siitä usein esiintyviä termejä tai fraaseja sekä ilmaisuja, jotka kannattaa kääntää käsin. Lisäksi esikäsittelyvaiheessa voidaan lisätä materiaaliin käännösjärjestelmää helpottavaa informaatiota ja korjata tekstissä esiintyviä kielellisiä puutteita.

2. Viritystyö

Järjestelmän kehitysasteesta riippuen voidaan tarvita viritystyötä, jonka aikana konekäännösjärjestelmää viritetään sovellutusalueeseen, asiakkaan tarpeisiin ja käännöstehtävään sopivaksi. Viritystyö vaatii usein asiantuntijoiden työpanosta. Tässä vaiheessa mm. analyysivaiheessa löytyneille vieraille sanoille annetaan käännösvastineet.

3. Jälkikäsittely

Viimeisessä vaiheessa mm. tarkistetaan käännöstulos ja palautetaan viesti lopulliseen muotoonsa (esim. valittuun tiedostomuotoon).

Ihmisten ja varsinkin asiantuntijoiden työaika on kallista. Eräänä konekäännösjärjestelmän hyvyyden mittana voidaankin pitää sen vaatimien ihmisresurssien määrää: mitä vähemmällä työllä hyväksyttävä käännös syntyy, sitä parempi järjestelmä.

3.1.3. Mukautumisongelmien yleisyys

Adaptiivisuusongelmat voidaan jakaa niiden yleisyyden mukaan. Ongelmien yleisyys vaikuttaa niiden ratkaisuun käytettävien menetelmien siirrettävyys- ja modulaarisuusvaatimuksiin.

Konekääntämisen yleiset mukautumisongelmat

Kaikki konekäännösjärjestelmät joutuvat esimerkiksi seuraavanlaisten ongelmien eteen, kun niitä aletaan käyttää tuotantokäytössä:

- asiakirjojen erilaiset tiedostomuodot ja niiden käsittelyongelmat [de Schaetzen, 1994],
- käännettävän tekstin joukossa olevat kuvat, taulukot yms. poikkeukselliset symbolit ja niiden erotteleminen,
- lähdekielisen tekstin syntaksin puutteellisuus ja sen jäsentämisen ongelmat, käännösyksiköiden erotteleminen, kielen monitulkintaisuus [Arnola, 1995a]. (Esim. *Putkien tulee olla puhtaita - putka* vai *putki* ?),

- käännössääntöjen ja käännössanaston puutteellisuus: luonnollisen kielen rikkaus vaatii aina suurta sääntömäärää [Hutchins, 1986].

Kielikohtaiset mukautumisongelmat

Lähde- ja kohdekielen välinen yhteys vaikuttaa muun muassa tarvittavaan analyysin syvyyteen, mistä seuraa vaatimuksia opetettavuudelle. Jos esimerkiksi käännettään saman kielen murteelta toiselle, ei analyysiä, eikä myöskään rakenteellisia muunnossääntöjä välttämättä tarvita. Toisaalta kieliperheistä toisiin käännettäessä tarvitaan syvempää analyysiä ja suurempaa tietokantaa.

Tietyt kielet, kuten aasialaiset kielet vaativat erikoismerkistöjä, mikä saattaa olla ongelma käännösjärjestelmälle. Merkistöjen koodaustavoista saattaa olla useita eri käytäntöjä.

Järjestelmäkohtaiset mukautumisongelmat

Valittu käännösperiaate ja järjestelmän arkkitehtuuri vaikuttavat yleensä varsin ratkaisevasti järjestelmän mukautumistarpeisiin ja -kykyyn. Esimerkiksi Kielikone Oy:n TranSmart-järjestelmän eräitä nykyisiä ongelmia ovat:

- järjestelmää kehitettäessä sen lingvistiseen rakenteeseen korjaustarkoituksessa tehdyt, jotakin huonosti kääntyvää kielellistä ilmiötä koskevat muutokset saattavat aiheuttaa **kokonaisuutena katsoen** vaikeasti havaittavia taantumia järjestelmän tuottamissa käännöksissä, vaikka korjausten lokaalisen seuraukset olisivatkin haluttuja.
- suomen kielen nominatiivisia jälkiattribuutteja saavien sanojen tunnistaminen on jäsentimelle vaikeaa ja perustuu lekseemien luettelointiin.
- uusien sanojen lisääminen morfologisen analysaattorin - Morfon - sanastoon saattaa huonontaa järjestelmän suorituskykyä sanojen monitulkitaisuuksien lisäännyttyä.
- virkkeentunnistin ei ole sääntöohjattu eikä siten helposti opetettava.

Sovellusalue-, käyttäjä- ja tekstikohtaiset ongelmat

Jollain sovellusalueilla sananmuotojen monitulkintaisuus saattaa tuottaa ongelmia monien mahdollisten tulkintojen takia. Ongelma ratkeaa, jos käyttäjä voi tehdä sanastussääntöjä, joiden laukeaminen riippuu sanan ympäristöstä.

Adaptiivinen järjestelmä ei hukkaa käyttäjän kerran tiettyyn tekstiin tekemiä korjauksia, kun teksti käännetään uudestaan.

3.1.4. TranSmartin käyttäjien tarpeet

TranSmartin käyttäjien päätarpeet ovat selvinneet melko hyvin vuosien prototyyppitestauksen avulla. Perustilanteessa käyttäjällä on asiakirja, josta hän haluaa kieleltään ja muotoiluiltaan kunnollisen käännöksen. Täysin koneellisesti hänen tarvettaan ei kuitenkaan vielä voida tyydyttää. Tämä tosiseikka pitää vielä ottaa huomioon ja pyrkiä käyttäjän käsityön minimoimiseen.

Kielikone Oy:n asiakkaan Nokia Telecommunicationsin päätarve liittyen TranSmartin adaptiivisuuteen painottuu käännösmuistiin: he haluavat kääntäjien tarkistamien käännösten pysyvän muuttumattomina ja uudelleen käännettäessä vain muuttuneiden osien kääntyvän käännössääntöjen avulla.

Käännetyssä asiakirjassa yksi suomenkielinen virke voi kääntyä useaksi englanninkieliseksi virkkeeksi tai toisinpäin. NTC:n haluaa mahdollisuuden tallettaa käännösmuistiin myös tällaisia vastaavuuksia. Käännösmuistin tulisi myös olla joustava koodien suhteen: jos tietokantaan on talletettu virke, jossa vain jokin koodi eroaa käännettävästä virkkeestä, pitäisi esimerkkikäännöstä pystyä kuitenkin hyödyntämään.

Source: Paina nappia ##Code##.
Target: Press button ##Code##.

Lisäksi NTC:n tarpeeseen liittyy sumea haku käännösmuistista: sieltä tulisi voida etsiä virkkeitä, jotka suunnilleen täyttävät hakuehdot. Käännösmuistista löytyvät virkkeet pitäisi myös voida erotella lopullisessa tekstissä esimerkiksi eri värillä. [Nuutila, 1995]

Käytäntö on osoittanut, että konekäännösprojekteissa käännettävä materiaali on laadultaan hyvin vaihtelevaa. Teksteissä on puutteita ja omia piirteitään. Kuitenkin materiaalista löytyy usein toistuviakin ilmiöitä, jotka kerta toisensa jälkeen joudutaan korjaamaan käsin. Olisi mielekästä, jos järjestelmälle voitaisiin opettaa erilaisia toimintatapoja, jolloin se osaisi seuraavalla käännöskerralla käsitellä nämä ilmiöt oikein.

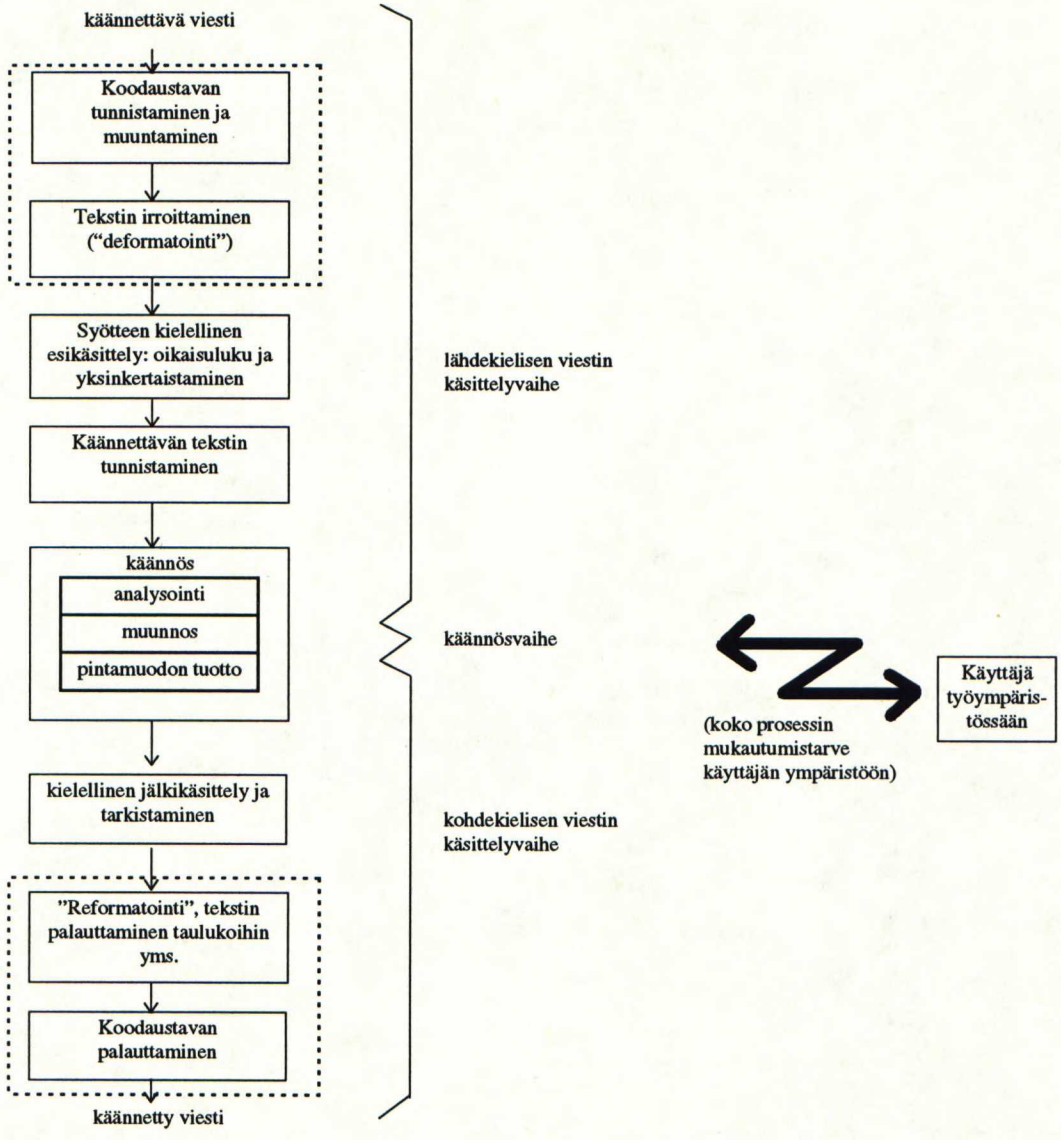
Asiakkaiden tekstit ovat suurimmaksi osaksi erilaisilla tekstinkäsittelyohjelmilla tuotettuja tiedostoja. Jos niissä piilevä muotoilu ei säily käännöksen aikana, pitää käännetty teksti muotoilla käsin. Tämä ei ole asiakkaiden mieleen. Niinpä käännösjärjestelmän tulee ottaa huomioon tavallisten tekstinkäsittelyohjelmien tarpeet.

3.2. Konekäännösjärjestelmän mukautumistarpeet

3.2.1. Staattinen ja dynaaminen mukautumiskyky

Käännettävän viestin analysoinnissa, muuntamisessa ja lopullisen käännöstuloksen tuottamisessa on useita erilaisia huomioon otettavia seikkoja ja suoritettavia tehtäviä. Kuvassa 3.2 esitetään prosessin ne tehtävät, joissa lähes kaikki nykyiset käännösjärjestelmät tekevät virheitä ja tarvitsevat ihmisen apua. Tästä syntyy käännösjärjestelmille mukautumistarpeita. Käytännössä vaiheet ovat usein iteratiivisia ja päällekkäisiä eikä kaikkia välttämättä tarvita.

Konekäännösjärjestelmän mukautumiskyky voidaan lisäksi jakaa kahteen tyyppiin: dynaamiseen ja staattiseen mukautumiskykyyn. **A. Dynaaminen mukautumiskyky** tarkoittaa käyttäjän mahdollisuutta vaikuttaa käännösprosessiin (opetettavuutta) ja järjestelmän automaattista sopeutumista käyttöympäristöönsä (oppivuutta) sen jälkeen, kun järjestelmä on toimitettu. Käännösmuisti, johon käyttäjä tai järjestelmä voi itse lisätä virkkeiden vakiokäännöksiä, on esimerkki dynaamisesta mukautumisesta. **B. Staattinen mukautumiskyky** on järjestelmän pysyvä ominaisuus, jota ei tarvitse virittää käytön aikana, esimerkiksi sen kyky ymmärtää tietyn ennaltamäärätyn teksturin tiedostomuotoa [Arnola, 1995a].



Kuva 3.2. Viestin käsittelytarpeet käännösprosessissa

3.2.2. Mukautumistarve lähdekielisen viestin suhteen

Viestin koodaustapa

Ensimmäinen vaatimus jonkin viestin konekääntämiselle on viestin oikea esitystapa. Esimerkiksi paperilla oleva asiakirja pitää muuttaa sähköiseen muotoon, levykkeillä sijaitsevat tiedostot pitää siirtää umpilevylle, pakatut asiakirjat pitää purkaa pakkausohjelmalla ja niiden tiedostomuoto pitää muuttaa sopivaksi, esimerkiksi SGML-muotoon. Saattaa olla, että joudutaan tekemään useita erilaisia muunnoksia ennen kuin oikea tiedon esitystapa saavutetaan.

Viestien koodaustavat sisältävät usein paljon muotoiluun liittyvää tietoa. On tärkeää, että käännöstuloksen kannalta olennaisen hyödyllistä informaatiota ei kadoteta tässä vaiheessa. Esimerkiksi tekstin otsikoiden perässä ei usein ole virkkeen loppua ilmaisevaa pistettä, mistä syystä ne voivat liimautua yhteen seuraavan virkkeen kanssa. Käännöstuloksen laatu paraneekin, jos otsikot voidaan tunnistaa niiden suuremman kirjasintyypin tai suuremman rivivälin perusteella.

Käännettäessä puhetta voidaan puheen painotus, äänensävy yms. seikat joutua muuntamaan sähköiseen muotoon, sanojen attribuuteiksi. Tämä on sinänsä jo varsin suuri ongelma.

Tekstin irrottaminen

Tekstin deformatoimisen päätavoite on tekstin irrottaminen dokumentista: sen kuvista, taulukoista ja listoista sekä tekstin kulun, palstoituksen yms. selvittäminen. Jotta alkuperäinen muoto voidaan myöhemmin palauttaa, pitää tämä deformatointitieto tallettaa kohdekielisen viestin lopullista tuottamista varten [Boitet, 1990]. Jotain käännettävän viestin muotoon liittyvää tietoa voidaan hyödyntää myös varsinaisessa käännösvaiheessa. Tämä tieto liitetään käännettävän tekstin attribuuteiksi käännösjärjestelmän tunnistamilla koodeilla. Esimerkiksi otsikot on syytä merkitä viimeistään tässä vaiheessa. Tähän vaiheeseen liittyy myös joidenkin käännösyksiköiden rajaaminen.

Asiakirjojen kuviin, taulukoihin yms. liittyvän tekstin ja muun tiedon erottelemisen ongelma on ratkaistu monissa tavallisimmissa tekstinkäsittelyohjelmissa. Niinpä tekstiä sisältävien kuvien, taulukoiden yms. sisäisistä esitysmuodoista teksti ja sen kulku (peräkkäiset kirjaimet ja sanat) löytyvät helposti. Ongelmia syntyykin yleensä vasta silloin kun asiakirjan tekijä on käyttänyt teksturistaan riippumattomia keinoja tekstin muotoilemiseen tai esimerkiksi kuva on rasterimuodossa (bitmap). Esimerkiksi välilyöntien tai sarkainmerkkien avulla tehdyt taulukot ovat ongelmallisia (kuva 3.3).

Miten teksti juoksee <sarkain> *Tämä on toinen sarake,*
taulukossa? <sarkain> *jossa on jo toinen virke.*

Virheellinen käännös TranSmartista:

How does the text run, this is the second column, in the table? in which there is already another sentence.

Kuva 3.3. Sarkainten käyttö palstoitukseen tai taulukointiin

Listojen tulkitseminen ja säilyttäminen voi olla ongelmallista, koska aina ei tiedetä, onko rivin loppu myös käännösyksikön loppu (kuva 3.4).

Muista tuoda kaupasta:
viisi kiloa voita
kuusi maitoa

Muodoltaan virheellinen käännös TranSmart-käännösjärjestelmällä:

Remember to bring from the trade five kilos of butter six milks

Kuva 3.4. Sisennykset ja listat

Seuraavassa (kuva 3.5) on ote TKK:n Laskentakeskuksen oppaasta Ohjelmointi Unix-ympäristössä. Tekstissä on ensin lista, jonka rivinvaihdot katkaisevat käännösyksikön. Heti seuraavassa kappaleessa on samannäköinen lista, mutta rivinvaihto ei enää katkaise käännösyksikköä. Ensimmäisen listan rivinvaihtojen tulisi säilyä käännöksessä muuttumattomina. (Tämän lisäksi käännössanasto on puutteellinen, mistä seuraa %%- ja **-merkkejä sekä vääriä sananvalintoja. Myös epätäydellisten virkkeiden kääntäminen tuottaa ongelmia.)

Atk-keskus tarjoaa asiakkaiden käyttöön seuraavat Unix-koneet:

IBM:n RS/6000-koneita, käyttöjärjestelmänä IBM:n Unix-toteutus Aix:

Vipunen (sähköpostiin ja muuhun viestintään)
Leka numeeriseen laskentaan
Cactus numeeriseen laskentaan (CSC:n kanssa yhteisomistus)
Cad (CAD- ja CAE-käyttöön, huoneessa U132)

HP:n työasemia, käyttöjärjestelmän HP:n Unix-toteutus HP-UX:

22 kpl HP9000/705-työasematietokoneita harmaasävynäytöllä ja 4 kpl kesällä-94 hankittuja HP9000/712 työasematietokoneita värinäytöllä Maarintalossa. Nämä tietokoneet on nimetty lk-hp-1, ...,lk-hp-26
HP9000-sarjan tehokkaita 3D-työasemia Freud, Marx, Engels, Jung huoneessa U132

Virheellinen käännös TranSmartista :

The ADP exchange offers the following Unix machines into the customers' use:

RS/6000 machines of IBM as operating system Unix realisation of IBM Aix:

Vipunen to the electricity post and to an other communication) sledge to digital calculation %%Cactus%% to digital calculation (with %%CSC%% a joint ownership) %%Cad%% (into CAD and CAE use in the room %%U132%%)

Workstations of HP, Unix realisation of the HP of the operating system HP %%UX%%:

*22 pieces of HP9000/705 **työasematietokone** on **harmaasävynäyttö** and 4 pieces in summer -94 acquired **työasematietokone** in the **Maarintalo** on the colour monitor HP9000/712. These computers lk hp have been named -1 a %%HP9000%% series of efficient ones, the %%3D%% workstations, in the room ...lk*
hp -26 Freud, Marx, Engels %%Jung%% %%U132%%

Kuva 3.5. Listojen tunnistamisen vaikeus

Rasterikuviin piilotettu käännettävä teksti on luonnollisesti suuri ongelma, koska kirjasintyypit voivat olla varsin vapaamuotoisia. OCR-ohjelmia voidaan käyttää tekstin tunnistamiseen rasterikuvista.

Merkistöt liittyvät yleensä viestin koodaustapaan eli ensimmäiseen mukautumistarpeeseen. On kuitenkin olemassa monia merkkejä, joiden käyttötarkoitus riippuu asiayhteydestä, kirjoittajasta, tyylistä tai muista seikoista. Yhtenä esimerkkinä on pieni o-kirjain, jota voidaan käyttää listoissa (kuva 3.6).

Ennen kuin lähdet, muista seuraavat asiat:
- *o leikkaa nurmikko*
- *o kastele kukat*

Virheellinen käännös TranSmartista:

Following before you leave, o cuts the flowers from others, the lawn o, water

Kuva 3.6. Onko o-merkki listan alku vai yksikirjaiminen koodi ?

Sananrajakaan ei aina ole selvä, jopa välilyöntiä voidaan käyttää keskellä sanaa. Tätä vahvistuskeinoa käytetään ainakin saksankielisessä kirjallisuudessa (kuva 3.7) [Hutchins, 1992].

Mitä t ä s s ä lukee ?

Virheellinen käännös TranSmartista:

T ä ä what does s s read ?

Kuva 3.7. Välilyöntejä keskellä sanaa

Eräs esimerkki monitulkintaisesta merkistä on myös alaviiva:

Tämä_on_tärkeää.

Tässä on koodi AUTO_1212:

Kuva 3.8. Alaviivan monitulkintaisuus

Mm. näiden esimerkkien perusteella nousee tarve tulkita näitä yksittäisiä merkkejä yhteysriippuvasti.

Oikaisuluku

Konekäännösjärjestelmät asettavat käännettävälle tekstille ihmiskääntäjää tiukempia oikeakielisyysvaatimuksia. Ennen konekääntämistä käännettävä viesti kannattaakin oikaisulukea. Oikeinkirjoituksen tarkistamiseen on olemassa valmiita ohjelmia ja niitä voidaan käyttää myös konekääntämisessä. Voi kuitenkin olla tarkoituksenmukaisempaa käyttää käännösjärjestelmän omia sanastoja, koska niiden avulla selviää tarkemmin tekstin soveltuvuus käytettyyn järjestelmään.

Kielen yksinkertaistaminen

Nykyiset käännösjärjestelmät toimivat parhaiten, kun niillä käännettävän tekstin tyyli noudattaa tiettyjä rajoja. Tyyliin vaikuttavat mm. virkkeiden keskimääräinen pituus sekä harvinaisten tai monimutkaisten ilmaisujen määrä. Liian monimutkaiset ilmaisut eivät yleensä käänny hyvin koneellisesti, niinpä ne kannattaa muuttaa yksinkertaisemmiksi, jos se on mahdollista. Tätä voidaan pitää ensimmäisenä askeleena kohti alikieliä.

Käännettävän tekstin tunnistaminen

Käännettävät tekstit voivat olla monikielisiä, joilloin niissä saattaa olla osia, joita käännösjärjestelmän ytimen ei tarvitse käsitellä. Esimerkiksi kaavat, erisnimet, vieraskieliset lainaukset yms. tulee erotella tekstistä ja varmistaa, että ne eivät muutu käännöksessä. Tekninen

dokumentaatio, joka yleensä sopii kielellisesti hyvin konekäännettäväksi, sisältää hyvin paljon erityyppisiä koodinomaisia osia. Atk-asiakirjojen ohjelmakoodi on eräs esimerkki tekstiosasta, jota ei saa kääntää:

```
Älä käännä seuraavia rivejä:

for (x=0; x<10; x++)
   printf("%i",x);
```

Kuva 3.9. Koodinomaiset osat tekstin seassa

Selkeät kieliosat voidaan erottaa automaattisesti vaikkapa kielten yleisten sanojen esiintymisen perusteella [Giguet, 1995]. Kaavoja ja muita ei-kielellisiä osuuksia voidaan yrittää tunnistaa vaikkapa erikoismerkkien avulla. Niissä esiintyviä 'sanoja' löytyy harvoin käännössanastoista. Kääntämättä jätettävien alueiden tunnistaminen voidaan perustaa myös tähän seikkaan.

Suurimmat ongelmat piilevät kuitenkin koodinomaisissa tekstiosioissa, jotka sisältävät myös kääntämistä vaativaa tekstiä. Esimerkiksi *tulosta()*-funktion nimi ehkä haluttaisiin kääntää, jotta ohjelman toiminta olisi selvä myös suomea osaamattomalle kohdekielistä asiakirjaa lukevalle henkilölle.

```
for (x=0; x<10; x++)
   tulosta ( x );
```

Kuva 3.10. Käännettävän tekstin tunnistaminen

Monet tekstinkäsittelyohjelmat mahdollistavat tekstiosioiden kielen määrittämisen. Jos käyttäjä on sen vaivautunut tekemään, tulisi tietoa myös pystyä hyödyntämään.

Sanastolliset monitulkintaisuudet

Sanojen monitulkintaisuuksien ratkominen saattaa olla vaikeaa koneellisesti. Esimerkiksi virkkeissä *Tuolla on paljon patoja* tai *Putkien tulee olla puhtaita* esiintyy homonyymejä (*pato/pata, putka/putki*), joiden disambiguointi ei onnistu ihmiseltäkään ilman asiayhteyttä.

Niinpä käännösjärjestelmät saattavat tarvita ihmisen apua monitulkintaisuuksien selvittämiseen. Tätä ongelmaa on myös mahdollista yrittää ratkaista tilastollisten menetelmien ja alakohtaisten sanastojen avulla.

Rakenteellinen monitulkintaisuudet

Rakenteellisiin monitulkintaisuuksiin törmätään jäsennysvaiheen aikana. Esimerkki puhekielen rakenteellisesta monitulkintaisuudesta on virkkeessä: *Voinko sovittaa noita housuja tuolla ikkunassa?* Onko *tuolla ikkunassa* housujen määre vai *sovittaa*-verbin adverbi?

Toki pelkkää kääntämistä varten ei aina ole edes välttämätöntä ratkaista kaikkia monitulkintaisuuksia. Kohdekielessa voi nimittäin olla sama monitulkintaisuus [Hutchins et al., 1992].

Semanttiset monitulkintaisuudet

Jos käännösjärjestelmä suorittaa lähdekielen semanttisen analysoinnin, saattaa tekstiin jäädä myös semanttisia monitulkintaisuuksia. Niiden ratkaisemiseen saatetaan tarvita käyttäjän apua.

3.2.3. Muunnosvaiheen mukautumistarpeet

Termit, sanat ja vakiokäännökset

Käyttäjät haluavat usein vaikuttaa käännösjärjestelmän tekemiin sananvalintoihin tai termien käännöksiin. Järjestelmän tulee tarjota tähän mahdollisuus. Suurin osa käännösjärjestelmien tekemistä virheistä liittyy yleensä vääriin sanavalintoihin. Käännösjärjestelmän tulisikin siis mahdollistaa sanatason käännössääntöjen lisääminen.

Usein on tarvetta myös kokonaisten ilmausten vakiokääntämiseen. Vaikkapa suomenkielisten ohjelmien valikoissa esiintyvä *Talleta nimellä...* ilmaistaan englanniksi *Save As...* eikä esim. *Save with name....*

Rakenteelliset käännössäännöt

Järjestelmän arkkitehtuuri vaikuttaa siihen, voidaanko käyttäjälle antaa mahdollisuus vaikuttaa rakenteellisiin käännössääntöihin, jos sellaisia tarvitaan. Ongelmana on riittävän helpon rajapinnan luominen lingvistiikkaan ja järjestelmän toteutukseen perehtymättömälle käyttäjälle. Sääntöpohjaisen käännösjärjestelmän rakenteelliset säännöt on usein helppo myös sekoittaa esimerkiksi väärillä, liian yleisillä säännöillä. Niinpä käyttäjälle ei tule antaa mahdollisuutta liian yleisten muutoksien tekemiseen.

Monille käyttäjille riittää mahdollisuus joidenkin 'muuttujien' lisäämiseen vakiokäännöksiin. Tällaisten käännöshahmojen tekeminen ei vaadi kovinkaan paljon kielellistä tietämystä.

Esimerkki käännöshahmosta voisi olla vaikkapa jo edellä esitetty:

Source: *Paina nappia ##Code##.*
Target: *Press button ##Code##.*

Muotoilujen periytyminen

Ehdoton vaatimus muunnosvaiheelle on muotoilujen siirtyminen lähdekielisiltä sanoilta kohdekielisille sanoille. Muotoilut siirtyvät luontevasti sanojen piirteinä.

3.2.4. Mukautumistarpeet kohdekielisen viestin käsittelyssä

Käännöksen kielellinen oikeellisuus

Kaikesta esikäsittelystä ja sanastollisesta kehityksestä huolimatta käännöstulokseen saattaa jäädä virheitä. Viestin jälkikäsittely on usein käsityötä. Jälkikäsittelyssä voi olla hyötyä kielenhuolto-ohjelmista. Tämän lisäksi voidaan etsiä tyypillisiä järjestelmän tekemiä virheitä.

Muotoilu

Esikäsittelyvaiheessa deformatoidun asiakirjan muotoilut ja sommittelu pitää palauttaa. Tehtävä on automatisoitavissa, jos deformatointi on tehty tätä toimenpidettä silmälläpitäen eikä tarpeellista tietoa ole kadotettu käännösprosessin aikana.

Käsityönä tehty kohdetekstin muotoilu kannattaa tehdä kaupallisilla tekstureilla.

Koodaustapa

Konekääntämisen viimeinen vaihe on lopullisen asiakkaalle toimitettavan käännöstuloksen tuottaminen. Tähän voi liittyä tiedostomuotojen muuttamista sekä asiakirjojen tulostamista, siirtämistä levykkeille. Toimenpide on vastakkainen ensimmäiselle käsittelyvaiheelle.

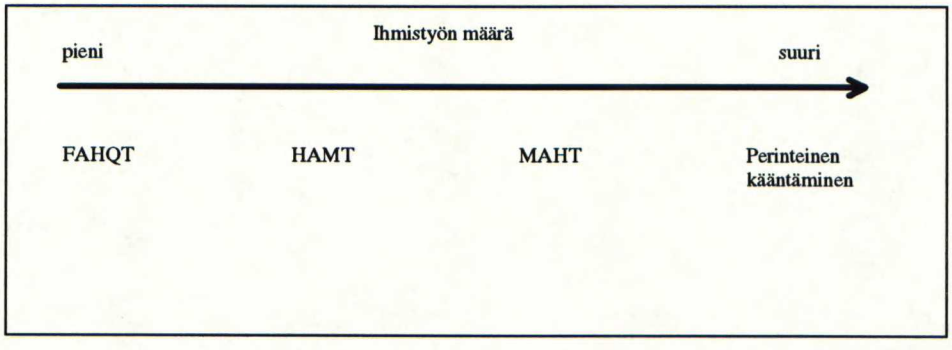
3.2.5. Mukautuminen käyttäjän työympäristöön

Sen lisäksi, että konekäännösjärjestelmän varsinaisten käännösvaiheiden tulee olla mukautuvia, myös koko järjestelmän käyttöliittymän tulee olla joustava ja käytettävä. Esimerkiksi monet kääntäjät ja tekstintuottajat ovat tottuneet käyttämään työssään tunnettuja tekstinkäsittely-ohjelmia. Parhaaseen tulokseen päästään, jos käyttäjän ei tarvitse jättää tuttua tekstinkäsittely-ohjelmaansa kääntämisen tai käännösjärjestelmän opettamisen ajaksi.

3.3. Esimerkkejä käännösjärjestelmien mukautumiskyvystä

3.3.1. Kielellinen mukautumiskyky

Koneen avustama ihmiskääntäjä tai ihmisen avustama konekäännin



Kuva 3.11. Kone- ja ihmiskääntäminen

Käännöstyön tekemiseen käytettävät menetelmät voidaan jakaa suoralle, jonka toisessa päässä on ihmisen kokonaan suorittama käännöstyö ja toisessa päässä täysin automatisoitu korkealaatuinen kääntäminen (FAHQT - Fully Automated High Quality Translation). Näiden ääripäiden välistä löytyy erilaisia hybridijärjestelmiä: ihmisavusteinen konekääntäminen (HAMT - Human-Aided Machine Translation) ja koneavusteinen ihmiskääntäminen (MAHT - Machine-Aided Human Translation) [Hutchins et al., 1992].

Yleiskielestä täysin virheetöntä käännöstulosta tuottavaa FAHQT-järjestelmää ei ole olemassa. Niinpä käytännössä monet käännösjärjestelmät perustuvat osittaiseen interaktioon käyttäjän kanssa. Näistä järjestelmistä käytetään mm. nimityksiä interaktiivinen (Interactive MT), keskusteluperusteinen (Dialogue-based MT) tai henkilökohtainen konekääntäminen (Personal MT).

Monet interaktiiviset käännösjärjestelmät ovat ns. kääntäjän työasemia. Tyypillisesti kääntäjän työasemat toimivat siten, että kääntäjä näkee samanaikaisesti näytöllä sekä lähde- että kohdekielisen tekstin. Kone kääntää virkkeen ja tarvittaessa tulostaa keskusteluikkunan, jossa kääntäjä voi valita käännöksen monien ehdotuksien joukosta. Kääntäjä voi halutessaan muokata tekstiä myös tavallisin tekstinkäsittelykomennoin. Kääntäjän työasemat saattavat tallentaa käsin käännetyn virkkeen ns. käännösmuistiin, josta sitä voidaan myöhemmin hyödyntää.

Käännösmuistit

Käytännössä käyttäjät saattavat jopa hylätä produktiivisesti käännöksiä tuottavan sääntöpohjaisen käännösjärjestelmän sen puutteiden takia, koska he joutuvat loppujen lopuksi kuitenkin aina korjaamaan käännöksen käsin. Käännösmuistiin perustuvat kääntäjän työasemat ovat kuitenkin kääntäjien mukaan osoittautuneet tehokkaiksi ja käyttökelpoisiksi. Nämä järjestelmät mahdollistavat vanhojen käännösten tallettamisen tietokantaan, josta niitä

voidaan tarvittaessa hakea. Kun monta kääntäjää jatkuvasti tallettaa työnsä yhteiseen tietovarastoon, sen hyöty ajan myötä kasvaa [Language International, 6/6, 1994].

Käännösmuistiin perustuvia kääntäjän apuvälineitä ovat mm. IBM:n Translation Manager, Tradosin Translator's Workbench, Starin Transit ja Eurolang Optimizer. Nämä eivät kuitenkaan ole täysverisiä käännösjärjestelmiä, koska niissä kääntäminen ei perustu lähdetekstin analysointiin. Niinpä uuden, neitseellisen tekstin kääntämisessä ei ainakaan pienestä käännösmuistista ole juurikaan hyötyä.

Myös käännösmuisteissa on ongelmansa. Kääntäjien tekemät kirjoitusvirheet tai käännösvirheet, jotka pääsevät muistiin muodostuvat helposti ongelmiksi. Virkkeen sisäisten muotoilujen periyttäminen on vaikeaa, jos muotoilut poikkeavat käännettävän ja talletetun virkkeen välillä.

Rajoitettuihin kieliin tai alikieliin nojautuvat järjestelmät

Rajoitettuihin tai alikieliin perustuvat järjestelmät edellyttävät, että käyttäjän antama syöte noudattaa tarkoin määriteltyjä rajoja - ne siis vaativat käyttäjältä mukautumiskykyä (!). Jos tämä on mahdollista, ovat järjestelmät melko käyttökelpoisia, koska ne eivät yleensä edellytä käyttäjän apua varsinaisessa käännösvaiheessa. Näin päästään lähelle ideaalista järjestelmää rajoitetussa ympäristössä.

Esimerkkiperusteinen konekääntäminen

Alunperin Makoto Nagaon esittämän **esimerkkiperusteisen** käännösjärjestelmän perusidea on yksinkertainen: käännetään lähdevirke matkimalla samantapaisten valmiiden esimerkkien käännöksiä. Oletetaan esimerkiksi, että entuudestaan tiedetään seuraavien virkkeiden käännökset.

Hän osti säkillisen omenoita. → *He bought a sack of apples*
Luin kirjan kansainvälisestä politiikasta. → *I read a book on international politics,*

Nyt voidaan kääntää virke *Hän osti kirjan kansainvälisestä politiikasta* yhdistämällä ensimmäisen virkkeen alkuosa eli subjekti ja predikoiva verbi toisen virkkeen loppuosan eli sen objektina toimivan substantiivirakenteen kanssa.

→ *He bought a book on international politics.*

Koska tekstisegmenttien käännökset riippuvat usein myös niiden ympäristöstä (context), osien valitseminen perustuu sekä niiden että niiden ympäristöjen samankaltaisuuteen.
[Sato et al., 1990]

Samantapaisia menetelmiä kutsutaan myös muistiperusteiseksi (Memory-Based MT) tai analogiaperusteiseksi (Analogy-Based MT) konekääntämiseksi.

Tämä on teoriassa varsin käyttökelpoinen ja joustava menetelmä konekäännösjärjestelmän perusperiaatteeksi. Menetelmä on pelkkiä käännösmuisteja tehokkaampi, koska se hyödyntää myös virkkeiden osia. Se mahdollistaa käännösjärjestelmän jatkuvan kehittämisen, osaa hyödyntää valmiita käännöksiä, eikä tarvitse erillistä sääntökieltä.

Toisaalta menetelmän soveltaminen käytännössä vaatii varsin paljon vaikeasti toteutettavia osakokonaisuuksia. Vaikein ongelma on vastinvirkkeiden jakaminen oikeankokoisiin osiin ja näiden käännösvastineiden löytäminen (ns. segmentation ja kohdentamis- eli alignment -ongelmat). Liian lyhyet osat eivät käänny samalla tavalla kaikissa yhteyksissä ja liian pitkiä ei esiinny tarpeeksi usein, jotta niitä voitaisiin riittävästi hyödyntää. Riittävän yleistä ja

toimintavarmaa sanatasolla toimivaa tekstin kohdentamisalgoritmia ei ole vielä löydetty [Ahrenberg, 1995].

Ihmiskääntäjät muuttavat helposti kääntäessään virkkeiden rakenteita niin paljon, että koneen on hyvin vaikea löytää oikeita vastinosia. Ihminen voi esimerkiksi kääntää *Miehellä ei ole rahaa* → *The man is moneyless*. Kun koneen pitää tasata sanatasolla nämä virkkeet, voi liian yleinen algoritmi saada sanan *raha*:n käännökseksi *moneyless*, mikä voi tuottaa jatkossa merkitykseltään aivan vääriä käännöksiä. Esimerkkiperusteisen käännösjärjestelmän opettamiseen tarvitaan siis suuri määrä oikeanlaisia valmiita käännöksiä eli oikea opetusjoukko (training set) tai erittäin hyvä tasausalgoritmi.

Nagaon esittämässä esimerkkiperusteisessä kääntämistekniikassa tarvitaan myös molempien kielten jäsennyspuut, jotta voidaan etsiä virkkeiden vastinosapuut. Se asettaa niinikään tiedon esitysmuodolle suuria vaatimuksia, koska käännösprosessin aikana joudutaan etsimään sopivia palasia potentiaalisesti hyvin suuresta tietomäärästä. Jos jäsennyspuut ovat käytössä, saadaan siitä apua myös tasausongelmaan [Watanabe, 1993].

Esimerkkiperusteisten järjestelmien tutkimus on vireää ainakin Japanissa ATR-projekteissa (Advanced Telecommunications Reseach) [Hutchins, 1992].

Tilastollinen konekääntäminen

Puhdas tilastollinen käännösjärjestelmä perustuu sanojen esiintymisen todennäköisyyteen valmiissa käännöksissä. IBM:n Candide-projektissa alkuperäiskieliset ja käännetyt virkkeet kohdennetaan ja virkkeiden sisällä oleville sanoille lasketaan todennäköisyys sille, kääntyykö sana kahdeksi tai yhdeksi sanaksi vai putoaako se kokonaan pois. Tämän perusteella saadaan aikaan käännössäännöt. Menetelmää voidaan parantaa lisäämällä esimerkiksi morfologinen analysointi järjestelmään.

3.3.2. Mukautuminen viestien esitysmuotoon

Ei-lingvististen, tekstin muoto- ja koodausseikoista sekä järjestelmän ympäristöriippuvuustekijöistä aiheutuvien ongelmien kattava ratkaiseminen on hieman työläämpää, koska niiden kirjo on laajempi. Tekstin typologiaa, tyylejä ja välimerkkien käyttöä on yleensäkin tutkittu melko vähän konekääntämisen yhteydessä [Hutchins et al., 1992]. Siemensin METAL-käännösjärjestelmässä on käytössä modulaarinen ratkaisu: erillinen SINIX-järjestelmä hoitaa esi- ja jälkikäsittelyvaiheet. Siihen kuuluu joukko erikoistuneita ohjelmia, jotka hoitavat tekstin irrottamisen diagrammeista, taulukoista, yms. Vastaavasti käännetyt tekstit palautetaan alkuperäiseen muotoonsa käännösprosessin jälkeen [Hutchins et al., 1992]. Käytännössä monissa muissa järjestelmissä tiedostojen muuttaminen ja siirtäminen hoidetaan usein käsin.

LIDIA-projektissa tavoitteena on **henkilökohtainen konekääntäminen** (Personal MT) ja mahdollisimman korkealaatuisen käännöstuloksen tuottaminen käyttäjälle, jolla ei ole erityistä lingvististä taustatietoa. Järjestelmä perustuu käyttäjän ja järjestelmän interaktioon aina asiakirjan typologian analysoinnista lähtien. [Boitet, 1990]

Käytettävimmät järjestelmät lukevat ja kirjoittavat useiden tekstinkäsittelyohjelmien omia tiedostoja. RTF (Rich Text Format) on kuitenkin eräs eniten käytettyjä tiedostomuotoja, koska useimmat teksturit tukevat tätä vapaasti käytettävää tiedostomuotoa [de Schaetzen, 1994]. SGML (Standard Generalized Markup Language eli ISO 8879) ei vielä ole levinnyt käytännön sovellutuksissa.

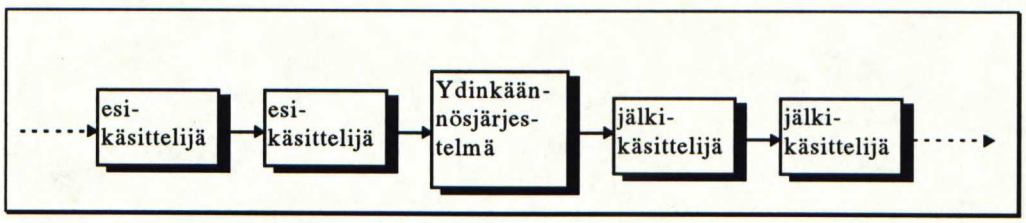
Useimmat nykyiset käännösjärjestelmät painottuvat asiakirjojen kääntämiseen. Kehityksen seuraava askel, puheen reaaliaikainen kääntäminen, on kuitenkin tutkimuksen kohteena

esimerkiksi saksa-englanti -järjestelmässä Verbmobilissa ja Japanin ATR Interpreting Telephony Research Laboratories:ssa. Puheen kääntäminen nostaa kuitenkin esiin suuren joukon uusia ongelmia, kuten puheen tunnistamisen ja syntetisoinnin sekä puheen rytmin, painotusten ja äänen sävyjen kääntämisen ongelmat [Kulikov, 1991]. Toisaalta kysyntä lienee suurempi, koska puheen kääntämisessä laatuvaatimukset ovat alhaisempia ja konekääntämisen nopeuden hyöty suurempi.

3.4. TranSmartin staattisen mukautumiskyvyn parantamismahdollisuuksia

3.4.1. Yleistä

Staattisen adaptiivisuuden ominaisuudet olisi kätevä toteuttaa erilaisten esi- ja jälkiprosessorien avulla, jotka voisivat tarvittaessa toimia yhdessä [Knight et al., 1994]:



Kuva 3.13. Staattisen adaptiivisuuden esi- ja jälkikäsittelijät.

Esikäsittelijät voivat joko toimia automaattisesti osana järjestelmää (esim. tekstitaulukon tunnistaja) tai käyttäjän erikseen käynnistäminä (esim. paperiasiakirjan lukeminen tunnistuslaitteella). Tällainen toteutus takaisi käsittelijöiden siirrettävyyden ja niiden toiminnan lokaalisuuden. Yleensä kannattaa pyrkiä käyttämään valmiita vakiotuotteita, koska ne ovat huokeita ja niiden ylläpitoon ei tarvitse käyttää omia resursseja.

3.4.2. Muotoilujen periyttäminen käännökseen - RTF-SGML-RTF-muunnin

Suurin osa konekäännettävistä asiakirjoista tehdään erilaisilla tekstureilla, jotka yleensä tallettavat tiedostonsa omassa esitysmuodossaan. Käyttäjän ympäristössä toimivan käännösjärjestelmän ehdoton vaatimus on, että sillä voi kääntää näitä asiakirjoja.

Luvussa 3.2 esiteltiin tekstien ongelmallisia sommitteluun liittyviä ominaisuuksia, kuten listoja ja kuvia. Jo se, että käännösjärjestelmä ymmärtää teksturien perusominaisuuksilla tehdyt muotoilut, riittää hyvän käännöstuloksen saavuttamiseen. Näin on ainakin olemassa yksi tapa, jota noudattamalla käyttäjä voi olla varma, että hänen asiakirjansa, listansa, kuvansa yms. kääntyvät oikein.

TranSmart pyrkii säilyttämään kääntämänsä asiakirjan SGML-muotoilut. Käytännössä esimerkiksi RTF olisi useimmille käyttäjille kuitenkin mielekkäämpi tiedostomuoto, koska heidän käyttämänsä tekstinkäsittelyohjelmat tukevat sitä [de Schaetzen, 1994] . Ongelmia saattaa kuitenkin syntyä RTF:n muuttamisessa SGML:ksi. SGML:ää ei ole suunniteltu niinkään muotoilun, vaan tekstin rakenteen kuvaamiseen. Varsinkin SGML:n muuttaminen takaisin RTF:ksi konekääntämisen jälkeen voi olla vaikeaa, koska käännösjärjestelmän tuottama SGML ei välttämättä ole puhdasta.

Toisaalta RTF voidaan konekäännösjärjestelmän tarpeita varten muuttaa melko helposti väliaikaisesti SGML:ksi siten, että vain teksti irroitetaan RTF-asiakirjasta. Muunnoksen helppous perustuu RTF:n ja SGML:n samankaltaisuuteen: RTF koostuu 7 bittisistä ASCII-merkeistä ja sen muotoilut koostuvat SGML:n tapaan alku- ja loppumerkeistä (kuva 3.14) [RTF-specification, 1994].

"{\b lihavoitu} sana"	= "**lihavoitu** sana"

Kuva 3.14. Lihavointi RTF-koodauksessa

Itse muotoiluihin ei pääsääntöisesti tarvitse ottaa kantaa, joten ne voidaan tallettaa RTF:nä SGML-merkintöjen sisään. Tämän lisäksi mm. 8-bittiset merkit ja SGML:n varatut merkit tulee muuntaa SGML:ksi. Siis esimerkiksi:

```
....
\par T\'e4m\'e4 on {\b lihavoitu}
\par Rasterikuva: {\pict\wmetafile8\picw847\pich1016\picwgoal480\pichgoal576
\picbmp\picbpp4
0000003000280000000000280000002800000030000000010004000000 0
000c003000000000000
...
000000000000000}
\par
```

Kuva 3.15. Kuva RTF-koodauksessa

[RTF-specification, 1994]

muutettuna MT-SGML:ksi:

```
...

T&auml;m&auml; on <RTF"{\b"> lihavoitu<\RTF "}">
Rasterikuva: <!RTF-ENTITY
"{\pict\wmetafile8\picw847\pich1016\picwgoal480\pichgoal576 \picbmp\picbpp4
0000003000280000000000280000002800000030000000010004000000 0
000c003000000000000
...
000000000000000">
....
```

Kuva 3.16. RTF MT-SGML:nä.

Toisin sanoen, RTF-muotoilut voidaan piilottaa SGML-merkintöjen sisään. Nyt asiakirjan muuntaminen takaisin RTF:ksi on helppoa: se onnistuu yksinkertaisella korvausoperaatiolla. Tekstiä sisältämättömät osiot voidaan piilottaa esim. <!RTF-ENTITY > -merkin sisään. Tietyt RTF-oliot, kuten taulukon sarakkeen vaihtumisen ilmaiseva \CELL, pitää muuttaa esim. <!RTF-TRUNIT >-merkiksi, joka katkaisee käännösyksikön.

Perusratkaisuun tarvitaan siis vain kolme SGML-merkintää. Koska usein on tarpeen tunnistaa myös muotoiluja ja rivien välien pituuksia, voidaan analyysin tarkkuutta tarpeen mukaan lisätä. Muunnosohjelmassa voidaankin ottaa huomioon konekääntämisen tarpeet ja muuntaa muotoiluja rakenteeksi. Ohjelma voi esimerkiksi lisätä virkkeen loppumerkin tekstiin aina kun sen tyyli vaihtuu (esim. Otsikosta Leipätekstiksi).

Näin syntyy melko vähällä vaivalla ohjelma, jonka avulla SGML-merkinnät tunteva ja säilyttävä käännösjärjestelmä saadaan askeleen lähemmäksi tavallista käyttäjää. RTF-SGML-muunninta voidaan käyttää myös muiden tiedostomuotojen kääntämiseen siten, että ne muutetaan ensin RTF-muotoon. Tätä tarkoitusta varten on olemassa valmiita muunnosohjelmia [Tietokone, joulukuu 1994].

3.4.3. Tekstin kulun selvittäminen

Jos halutaan tukea myös muita teksteissä käytettyjä muotoilukeinoja, on edessä laaja työsarka. Ihmisten käyttämiä tekstin tehokeinoja on varsin runsaasti ja niiden käyttö riippuu asiayhteydestä. Joitain osaongelmia on kuitenkin mahdollista ratkaista.

Kuten luvussa 3.2 esitettiin, esimerkiksi pelkästään ASCII-merkkejä käyttäen tehdyt taulukot, kaavat yms. oliot voivat tuottaa ongelmia käännösjärjestelmille:

Tämä on <sarkain> *Tässä on*
taulukko. <sarkain> *toinen sarake.*

Samoin mm. käännösyksiköiden rajojen tunnistaminen tuottaa vaikeuksia.

Tämäntyyppisten yksiköiden tunnistamista varten olisi tarkoituksenmukaista olla selkeä, monitasoinen toteutustapa. Eräs mahdollisuus olisi käyttää Lingvististä Säännöllistä Lauseketta, joka tavanomaisten villikorttimerkkien lisäksi tuntisi tekstin kielellisiä ominaisuuksia. Näitä ominaisuuksia voisivat olla sana, numero, sanaluokka yms. Tunnistamisen jälkeen oliot olisi muutettava sopivaan muotoon konekäännöksen ajaksi. Kääntämisen jälkeen taulukot yms. pitäisi palauttaa alkuperäiseen muotoonsa.

Esimerkiksi taulukot pitäisi linearisoida käännöksen ajaksi ja merkitä palauttamista varten

<Taulukko <parametrit> >
Tämä on taulukko
<sarake>
Tässä on toinen sarake
<\Taulukko>

Tällaisten hahmojen tunnistajan tulisi olla opetettava, koska yleistä ja vedenpitävää taulukon tai kaavan määritelmää on vaikea löytää. Hahmontunnistaminen ja -muuntaminen kannattaa toteuttaa yhdessä toimivien esi- ja jälkikäsittelijöiden avulla: esimerkiksi taulukon tunnistaja tunnistaa ja linearisoi taulukon, sitä vastaava jälkikäsittelijä palauttaa käännetyn taulukon.

3.4.4. Lähdekielisen tekstin oikeakielisyyden tarkistaminen

Oikeakielisyyden tarkistamiseen voidaan käyttää valmiita kielenhuolto-ohjelmia. Jos käännösjärjestelmässä on erillinen jäsennin, myös sitä voidaan käyttää kielenhuolto-ohjelman pohjana. Näin varmistetaan tekstin soveltuvuus käytettyyn järjestelmään. Vaikkei käännösjärjestelmän jäsennin olisikaan irroitettavissa, ainakin muunnossanastoja kannattaa hyödyntää oikeinkirjoituksen tarkistuksessa. Sanastojen käytön avulla voidaan varmistaa, että tekstin sanat ovat järjestelmän tunnistamia [Boitet, 1990].

Ajan myötä tyypillisiä virheitä ja niiden korjauksia voidaan kerätä tietokantaan, jolloin on mahdollista tehdä korjaukset automaattisesti. Joissain nykyisissä tekstinkäsittelyohjelmissa on ns. AutoCorrect-toiminto, joka korjaa tavallisimpia oikeinkirjoitusvirheitä tekstiä kirjoitettaessa. Jos tällaista toimintoa ei ole käytettävissä, voidaan automaattinen korjaaminen tehdä etsi-ja-korvaa -toiminnolla.

3.5. TranSmartin opetettavuus osajärjestelmäkohtaisesti

Kaikkien sääntöperusteisten käännösjärjestelmien suunnittelijat ovat joutuneet jossain vaiheessa ottamaan huomioon järjestelmän opetettavuuden, koska he ovat ainakin itse joutuneet opettamaan järjestelmää.

Sääntöperusteinen käännösjärjestelmä koostuu monista osajärjestelmistä, jotka tarvitsevat erilaisia sanastoja ja säännöstöjä toimiakseen. Eräs mahdollisuus opetettavuuden parantamiseksi on antaa käyttäjän muokata näitä säännöstöjä.

Toteutukseltaan melko helppo ratkaisu olisi TranSmartin nykyiseen arkkitehtuuriin ja toimintaan kiinteämmin liitetty opetettavuusrajapinta. Järjestelmä sisältää nykyisellään joukon erilaisia sanastoja ja säännöstöjä, joissa määritellään tietyn tyyppiset sanat tai sanojen käännökset. Näiden listojen päivitettävyyttä ja ylläpitoa parantamalla olisi mahdollista lisätä järjestelmän adaptiivisuutta. Käytännössä tämä tarkoittaisi ainakin selkeän opettamiseen keskittyneen rajapinnan luomista.

Opetussanastoja voisivat olla mm. **lähdekielinen sanasto, vakiokäännettävien virkkeiden lista**, käyttäjän **käännössanasto, lista nominatiivisen jälkiattribuutin saavista sanoista, pisteellisten lyhenteiden lista** ja sanojen **disambiguointilista**. Nämä voitaisiin yhdistää yhdeksi sanastoksi, joka voitaisiin kääntää binäärimuotoon tehokkuussyistä.

Käyttäjälle tulee luoda käyttöliittymä, jonka avulla hän voi opettaa järjestelmälle esimerkiksi yleisten mallien tai metasääntöjen mukaisia sääntöjä. Käyttäjän antamien sääntöjen ilmaisuvoimaa on syytä rajoittaa, koska häneltä ei voi aina edellyttää kielitieteilijän tasoisia tietoja kääntämisestä.

Lähdekielisten sanojen opettaminen

Kielikone Oy:n WinMorfo-oikaisulukuohjelmassa on valmis käyttöliittymä, jolla käyttäjä voi lisätä suomenkielisiä sanoja järjestelmään. Tätä voidaan käyttää myös konekäännös-järjestelmässä, koska sanastot ovat yhteensopivia.

WinMorfosta voidaan helposti erotella erillinen ajettava tiedosto, jolla sanastoja voi ylläpitää.

Lauseentunnistimen opetettavuus

Varsinainen lauseentunnistaminen tehdään nykyään C-kielisessä ohjelmassa. Käyttäjän ei ole mahdollista opettaa sitä. Lauseentunnistin tarjoaa kuitenkin mahdollisuuden tuoda ulkopuolista tietoa siihen <*trunit*>-merkkien avulla. Tämän ansiosta voidaan joko tekstikohtaisesti ilmaista virkerajoja tai rakentaa erillinen esikäsittelijä, joka automaattisesti lisää rajoja.

Jäsentimen opetettavuus

Jäsentimen opetettavuuden parantaminen liittyy lähinnä tiettyjen avointen sanajoukkojen ylläpitomahdollisuuteen. Esimerkiksi nominatiivisia jälkiattribuutteja saavat sanat ovat sellaisia, jotka jäsentimen tulisi tietää ja siksi niiden joukkoa tulisi voida kasvattaa (esim. *mallissa Exclusive*).

Käännössanastojen opetettavuus

Melko yksinkertaisten sanastosääntöjen lisäämistä varten Kielikoneessa on kehitetty Xteelt-ohjelma. Xteeltillä voi antaa mm. sanan käännöksen ja jotain tietoa sanan konteksista. Xteeltiä voidaan käyttää pohjana sanastotyökalun rakentamiseen.

Muut opetussanastot

Muita jo käytössä olevia sanastoja ovat mm. virkkeentunnistimen käyttämät listat pisteellisistä lyhenteistä tai sanoista, joiden kirjoitusasu vaikuttaa niiden tulkintaan (esim. *SOTURI* saattaa tarkoittaa erisnimeä, kun taas *soturi* on normaali appellatiivi).

3.6. TranSmartin opetettavuus älykkään käännösmuistin avulla

Käännösmuisti on tietokanta, johon on talletettu käännöksiä ja josta erilaisilla hakuehdoilla voidaan hakea virkkeitä ja niiden käännöksiä. Käännösmuistia käytettäessä hyödynnetään vanhoja samanlaisten virkkeiden käännöksia. Yleensä käännösmuistit perustuvat pelkkään merkkijonovertailuun tai sumeaan merkkijonovertailuun [EAGLES, 1994]. Niissä verrataan lähdekielistä tekstisegmenttiä tietokantaan talletettuihin tekstisegmentteihin ja kun löydetään lähimpänä oleva esimerkki, palautetaan sen käännös. Haun mahdollinen sumeus perustuu siis lähdekielisen tekstin pintamuotoon ja etäisyys määritellään usein tekstinkäsittelykomentojen määränä (edit-distance). Ongelmana tällaisessa lähestymistavassa on se, että vaikka lähdekieliset käännösyksiköiden pintamuodot muistuttavat toisiaan, niiden käännökset saattavat olla erilaisia.

Dependenssijäsennystä hyväksi käyttäen on mahdollista rakentaa **älykäs käännösmuisti**, jossa käännösten haku perustuisi jäsennettyihin tekstisegmentteihin, ei tekstin pintamuotoon. Tämän ansiosta järjestelmä huomaisi esimerkiksi seuraavien merkitykseltään ja rakenteeltaan samanlaisten, mutta sanajärjestykseltään ja pintamuodoltaan erilaisten virkkeiden samankaltaisuuden:

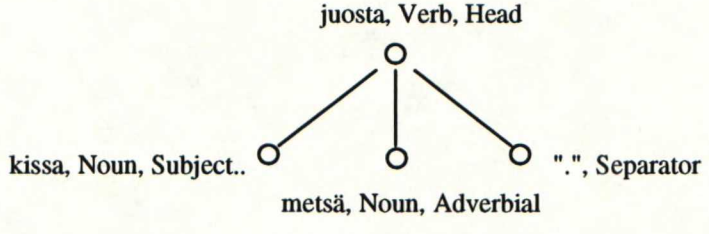
Matti oli täällä eilen. ←→ *Matti oli eilen täällä.*

Kuva 3.17. Pintamuodon ja semantiikan ero

Haettaessa tietoa älykkäästä käännösmuistista olisi myös mahdollista toteuttaa sumea haku, joka osaisi tarpeen mukaan karsia hakuehtoja ja hakea lingvistisesti toisiaan lähellä olevia ilmauksia.

Lingvistiseen käännösmuistiin talletettaisiin virkkeiden tai niiden osien jäsennyspuita. Kuvassa 3.18. on esimerkki jäsennyspuusta.

Kissa juoksee metsässä:



Kuva 3.18. Jäsennyspuu

Tällaiselle käännösmuistille löytyy ainakin **kaksi käyttötarkoitusta**: sitä voitaisiin käyttää interaktiivisesti esimerkkikäännöksien hakemiseen. Näistä käyttäjä voisi valita mieleisensä ja muokata sitä halutessaan. Toinen käyttökohde olisi konekäännösjärjestelmän osana. Tekstisegmenttien käännös voitaisiin valita käännösmuistista, jos käännettävän ja muistiin talletetun virkkeen **etäisyys** olisi riittävän pieni.

Käännösmuisti ratkaisisi monta käytännön adaptiivisuuden ongelmaa TranSmart-järjestelmässä, koska käyttäjille näyttää usein riittävän, että kerran korjatut käännösvirheet eivät uusiudu. Lisäksi sen käyttäminen olisi helppoa, koska erillistä opettamiskieltä ei tarvita.

Käännösmuistin laajennuksena olisi mahdollistaa yhdistää käännösmuistin ja konekäännösjärjestelmän säännöstön tietämys ja kääntää virkkeiden osa käännösmuistin ja osa käännössääntöjen avulla. Näin esimerkiksi virke *Koira juoksee metsässä* voitaisiin kääntää virkkeen *Kissa juoksee kadulla* käännöksen sekä sanojen *koira* ja *metsä* käännökseen avulla.

Eräänä lingvistisen käännösmuistin ongelmana on riippuvuus jäsentimestä. Luonnollisen kielen jäsentäminen on vaikea ongelma eikä TranSmartin jäsennin osaa jäsentää oikein kaikkia virkkeitä. Tällä hetkellä jäsennin jäsentää täysin oikein 80-90 % virkkeistä tekstityypistä riippuen [Arnola, 1995a]. Jos jäsennys on virheellinen, käännösmuisti voi antaa vääriä tuloksia. Jos jäsennin päivitetään, eivät vanhat käännösmuistiin talletetun virkkeet enää välttämättä löydy käännösmuistista. Saatavilla ei myöskään ole kuin suomen kielen dependenssijäsentäjä. Älykäs käännösmuisti olisikin yksisuuntainen: vain suomenkielinen virke voitaisiin etsiä sumeasti jäsennykseen perustuen. Muunkielisten virkkeiden sumeaa hakua varten tarvittaisiin merkkijonon vertailuun perustuva menetelmä.

Kirjallisuudessa puhutaan usein esimerkkiperusteisesta konekääntämisestä, kun viitataan tämän tapaiseen käännösmalliin [Piperidis et al., 1994]. Esimerkkiperusteisen kääntämisen ja käännösmuistin ero on siinä, että esimerkkiperusteisessa käännöstekniikassa voidaan käyttää hyväksi myös esimerkkikäännösten osia, eikä vain kokonaisia tietokantaan talletettuja käännösyksiköitä. Tällöin tarkastellaan myös käännettävien tekstiosioiden ympäristöjä ja niiden etäisyyksiä. Näyttää kuitenkin turhalta tehdä eroa käännösmuistin ja esimerkkiperusteisen käännöstekniikan välillä: molemmissa tavoitteena on löytää mahdollisimman hyvä käännösvastine esimerkkikäännöksien joukosta. Korkeintaan käännettävän tekstisegmentin koko saattaa vaihdella. Monet esimerkkiperusteisen käännösjärjestelmän ominaisuuksista tukevat adaptiivisuutta ja parantavat käännöslaatua:

- Käännöslaatu paranee, koska opettaminen on helppoa [Furuse et al., 1992].
- Stabiilisuus paranee, koska käännöstulokset pysyvät paremmin muuttumattomina järjestelmän kehittyessä.
- Joustavuus parantuu, koska lähdekielisen tekstin oikeakielisyysvaatimukset vähenevät [Furuse et al., 1992].
- Esimerkkiperusteinen käännösjärjestelmä soveltuu myös hyvin puhutun kielen kääntämiseen, koska virkkeet ovat siinä lyhyitä ja syntaksi vapaamuotoista [Furuse et al., 1992].
- Semanttisen tason saavuttaminen helpompaa [Yasuhara, 1993] (merkityksen opettaminen koneelle käsin on liian suuri tehtävä, joskin se on tavoitteena Dough Lenat'n johtamassa CYC-projektissa)
- Esimerkkiperusteiset järjestelmät on myös yleensä helppo virittää eri kielipareille.

3.7. Joitain menetelmiä automaattisen mukautumiskyvyn parantamiseen

Adaptiivinen käännösjärjestelmä osaa esimerkiksi tehdä yleistyksiä käyttäjän antaman tiedon perusteella. Tällainen järjestelmä osaa esimerkiksi tehdä yleistyksiä käyttäjän antaman tiedon perusteella. Toteutuksen liittyy aina jonkinasteinen heuristisuus. Esimerkiksi älykäs käännösmuisti tarjoaisi hyvän oppimismateriaalin (training set), jonka rakenteista olisi mahdollista tehdä yleistyksiä.

Käännettävän tekstin tunnistaminen

Tekstissä olevien vieraskielisten lainausten yms. automaattinen tunnistaminen voi perustua kieliopillisiin sanoihin ja merkistöön. Näin päästään virketasolla jopa yli 99%:n tarkkuuteen [Giguet, 1995]. Samaan tyyliin voidaan tunnistaa myös esim. ohjelmakoodi ja kaavat niiden erikoismerkkien ja varattujen sanojen avulla.

Erisnimien tunnistaminen

Erisnimien tunnistaminen ja merkitseminen on tärkeää oikean jäsennyksen takaamiseksi. Ellei niitä tunnisteta nimiksi, jotka voivat taipua, ne pitää olettaa koodeiksi. Nimien tunnistaminen on käännösprosessista irrallinen ongelma, joka voidaan ratkaista erillisen esikäsittelijän avulla. Erisnimistä voi olla valmis päivitettävä sanasto tai niiden tunnistaminen voi perustua esimerkiksi harvinaisten kirjaimien ja kirjainyhdistelmien esiintymiseen (Kazagstan) [Arnola, 1995a].

Tuntemattomia erisnimiä voidaan tunnistaa myös niiden esiintymispaikan ympäristön mukaan [Wolinski et al., 1995]. Tekstistä voidaan etsiä erilaisia hahmoja. Esimerkiksi:

<tunnettu erisnimi> <tunnistamaton merkkijono, iso alkukirjain>	(esim. *Pekka Sundgren*)
<tunnistamaton merkkijono> *Oy*	(esim. *Anor Oy*)

Tätä menetelmää voidaan käyttää myös erisnimen semanttiseen luokitteluun (yrityksen nimi, henkilön nimi yms.).

Artikkelien lisääminen tekstiin tilastollisiin menetelmiin perustuen

Tarkastelemalla suuria kielikorpuksia voidaan erilaisten ilmiöiden esiintymisfrekvenssejä tutkimalla selvittää niiden välisiä vuorovaikutuksia ja suhteita. Englanninkielisiä korpuksia on saatavilla hyvin paljon.

Esimerkiksi englannin kielen artikkelien (*the, a, an*) lisääminen tekstiin olisi sopiva ongelma ratkaistavaksi tällä tavalla. Knight et al. (1994) ovat kehittäneet tällaisen järjestelmän japani-englanti käännösjärjestelmää varten. Kuten suomi myös japani on artikkeliton kieli, mistä aiheutuu ongelmia artikkelilliseen kieleen kuten englantiin käännettäessä.

3.8. Yhteenveto: käyttäjän ehdoilla toimiva käännösjärjestelmä

Käytäntö on osoittanut, ettei ole tarkoituksenmukaista yrittää kerralla ratkaista kaikkia konekääntämisen kielellisiä ongelmia. Ehkä parempi ratkaisu olisikin mukautuva käännösjärjestelmä, jota käyttäjät voisivat itse kehittää ja virittää omiin tarpeisiinsa soveltuvaksi. Käännösjärjestelmän tärkeiksi ominaisuuksiksi nousevat tällöin sen kyky vastaanottaa uutta tietoa ja sen käytettävyys.

Mukautumiskyky

Mukautuva käännösjärjestelmä antaa käyttäjälle mahdollisuuden opettaa sille kielellistä informaatiota. Koska varsinaisten käännössääntöjen opettaminen on käyttäjille turhan vaikeaa, voitaisiin mahdollistaa yksinkertaisten sanastokäännössääntöjen lisääminen sekä kokonaisten lauseiden opettaminen.

Oletusarvoisesti opettavuuden tulisi toimia todellisten tekstien pohjalta: tarkistettaessa koneen kääntämää tekstiä virheet korjataan siten, että samalla lisätään korjatut sanat järjestelmän sanastoihin. Jos kyse on rakenteellisista puutteista, lisätään virke käännösmuistiin.

Käytettävyys

Mukautuvan käännösjärjestelmän tärkeimpiin tehtäviin kuuluu ydinkäännösjärjestelmän piilottaminen tarjoamalla siihen käyttöliittymä. Niinpä sen käytettävyys on tärkeää. Mukautuvan käännösjärjestelmän tulee toimia käyttäjän ympäristössä. Laajalle levinneet käyttöjärjestelmät kuten esim. MS-Windows-ympäristö tekstureineen ovat tyypillisiä käyttöympäristöjä.

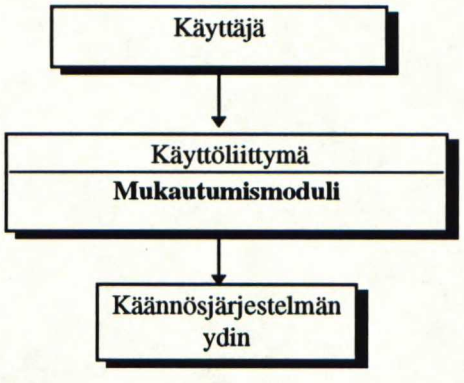
Esimerkiksi kolmelle eniten käytetylle tekstinkäsittelyohjelmalle, MS Word for Windowsille, WordPerfectille ja Ami Prolle (/Word Pro:lle), voitaisiin rakentaa joukko työkaluja, joilla TranSmartia käytettäisiin ja opetettaisiin.

Tyypillinen istunto voisi olla seuraavanlainen:

1. Käyttäjä lataa WinWord 7.0 -teksturiinsa käännettävän suomenkielisen tekstin ja valitsee *Työkalut/TranSmart/SU -> EN.*
2. Makro avaa teksturiin toisen ikkunan, johon käännettävä teksti syötetään mieluiten tausta-ajona, esimerkiksi kappale tai virke kerrallaan. Tiedostomuotona käytetään RTF:ää.
3. Käyttäjä huomaa virheellisesti kääntyneen sanan.
4. Hän maalaa sen ja valitsee *Työkalut/TranSmart/Lisää sana.*
5. Ruudulle ilmestyy valintaikkuna, jonka avulla sana ja sen käännös opetetaan järjestelmälle.
6. Ikkuna mukautuu valintojen ja olemassaolevan sanaston mukaan.
7. Käyttäjä käy tekstin läpi korjaten sanastopuutteet.
8. Lopuksi hän kääntää sen uudestaan.
9. Loput korjaukset käyttäjä tekee asiakirjaan käsin.
10. Ennen istunnon loppua hän vielä lisää virkkeet käännösmuistiin komennolla *Työkalut/TranSmart/Käännösmuisti.*

Modulaarisuus

Monet mukautuvan käännösjärjestelmän ominaisuudet ovat ydinkäännösjärjestelmästä irrallisia. Niinpä luontevin toteutus olisi erillinen osajärjestelmä, joka tarvittaessa kutsuu ydinkäännösjärjestelmää:



Kuva 3.19. Mukautumiskyvyn erottaminen

Näin siis voitaisiin piilottaa ydinjärjestelmä pehmeämmän rajapinnan taakse. Samalla saataisiin myös siirrettävä arkkitehtuuri - ydinkäännösprosessi voitaisiin tarvittaessa vaihtaa esim. toiselle kieliparille. Tällainen arkkitehtuuri tukisi luonnostaan mm. oppimista ja mahdollistaisi erilaisten käännösvaihtoehtojen generoinnin.

Tämän lisäksi mukautumiskyky asettaa vaatimuksia myös ydinkäännösjärjestelmälle. Sen pitää voida käyttää hyväkseen käyttäjän antamia lisäyssanakirjoja yms.

TranSmart on nykyään koko arkkitehtuuriltaan varsin vahvasti virkeorientoitunut. Jos siitä haluttaisiin tehdä 'aliohjelma', pitäisi ohjelmiston ylin taso kirjoittaa uudestaan.

Staattisen mukautumiskyvyn ominaisuudet kuten RTF-SGML-RTF-muunnin voidaan toteuttaa erillisinä ohjelmina. Älykäs käännösmuisti pitää toteuttaa ydinjärjestelmään, koska sen edellytyksenä on jäsennetty virke. TranSmart mahdollistaa useiden sanastojen käytön, joten käyttäjän omat lisäyssanastot eivät tuota ongelmia.

Järjestelmä ei myöskään mahdollista kaiken analysointia helpottavan tiedon tuomista ulkopuolelta käännösprosessiin. Jos haluttaisiin esimerkiksi merkitä etukäteen käsiteltävän tekstin erisnimet, pitäisi tätä varten järjestelmän toteutusta muuttaa.

Tehokkuus

Varsinkin käännösmuisti vaatii suuren määrän vastinvirkepareja, joten sen nopeus on tärkeää. Ihmisresurssien lisäksi myös aika saattaa joillain sovellusalueilla olla kallis resurssi.

4. Älykkään käännösmuistin rakenne

4.1. Ominaisuuksia

Käännösjärjestelmän adaptiivisuutta parantavista piirteistä toteutetaan tässä työssä älykäs käännösmuisti. Sen konekäännösjärjestelmän käytettävyyttä lisäävä vaikutus on selvimmin näkyvillä. Käännösmuistin laajennusmahdollisuudet ja käyttökohteet ovat niinikään selvät. Käännösmuisti on myös asiakkaiden toivelistalla.

Tavoitteena on rakentaa käännösmuisti, joka kykenee löytämään mahdollisimman paljon hyviä käännösehdotuksia. Kuten edellä on kerrottu, älykkäässä käännösmuistissa käännösten haku perustuu virkkeiden jäsennyspuihin, ei tekstin pintamuotoon. Näin hausta saadaan parametrisoitava ja kielellisesti tarkempi. Jäsentämisen ansiosta järjestelmä huomaa esimerkiksi seuraavien rakenteeltaan samanlaisten, mutta sanajärjestykseltään ja pintamuodoltaan erilaisten virkkeiden samankaltaisuuden:

Matti kävi Turussa eilen. ⟷ *Ville kävi eilen Porissa.*

Älykkäälle käännösmuistille löytyy kaksi **käyttötarkoitusta**: interaktiivinen esimerkkikäännöksien hakeminen ja esimerkkiperusteinen automaattinen konekääntäminen.

Ehkä suurin hyöty kielellisestä jäsentämisestä saadaan, kun käännösmuistiin yhdistetään **käännösmuokkain**, joka muokkaa käännösehdotuksena annettavaa kohdekielistä virkettä paremmaksi, jos lähdekielistä virkettä ei täsmällisesti löydy tietokannasta. Jos esim. tiedetään, että *Matti kävi Turussa eilen* kääntyy

Matti visited Turku yesterday,

voidaan tästä käännöksestä produktiivisesti muokata seuraava virke:

Ville visited Pori yesterday,

Tämä on jo hyväksyttävä käännös toiselle esimerkkivirkkeelle. Olemassaolevissa käännösmuisteissa ei tiettävästi ole tällaista ominaisuutta. Yleensä ne osaavat muuttaa korkeintaan eroavat numerot käännettävää virkettä vastaaviksi [Arnola, 1995a].

Käännösmuisti on jaettu osajärjestelmiin tietovuotekniikkaa (DF-diagram) käyttäen [Pressman, 1992]. Tietorakenne on kuvattu ER-kaavion avulla.

4.2. Tietovuokaavio

Kuva 4.1. (DFD 0, 0-tason tietovuokaavio) Käännösmuistin tietovuokaavio

Käännösmuistin palveluilla voi olla kaksi hyödyntäjää: sekä käyttäjä suoraan, että konekäännösjärjestelmä TranSmart. Niinpä muistin tulee tyydyttää kaksi erilaista tarvetta: käyttäjän tarve hakea joukko lähellä olevia käännösehdotuksia ja käännösjärjestelmän tarve hakea juuri oikea käännös, jos sellainen löytyy. Käännösjärjestelmä ei osaa arvioida annetun esimerkkikäännöksen hyvyyttä, siksi sille kelpaa vain oikea käännös.

Käyttäjä tarvitsee järjestelmää kääntäessään tekstiä. Hän haluaa käännösmuistista yhden tai useita käännösehdotuksia. Haun nopeus ole kovin aikakriittinen operaatio, koska ihminen tekee samalla käännöstyötään. Järjestelmä voi etsiä useampia käännösehdotuksia sekä järjestää ja analysoida niitä tarkemmin.

TranSmart tarvitsee nopeasti etsityn käännösyksikön oikean käännöksen. Käännöksessä ei saa olla vääriä sanoja tms. Nopeus on tärkeää varsinkin, jos käännösmuistia käytetään myös virkkeen osasten tallettamiseen, koska tällöin joudutaan tarkistamaan käännösmuisti yhden virkkeen monien osapuiden osalta.

4.2.1. Tietovirrat ihmiskääntäjän apuna

Käyttäjän syöte

Käyttäjän syötteenä voi olla:

- 1.1. Talletettava tekstipari
  eli suomen ja englannin kieliset asiakirjat, joiden tiedetään olevan toistensa käännöksiä.
- 1.2. Käännösehdotuksen hakupyyntö. Hakupyyntö on virke tai sen osa.
- 1.3. Tietokantakyselyt
  Näihin kyselyihin kuuluvat tietokantojen perustoiminnot kuten poisto-, listaus-, muokkaus- yms. pyynnöt.

Käyttäjän saama vaste

Vasteena käyttäjä saa:

- 2.1. Lista kyselyä lähellä olevista käännösehdotuksista.
  Tämä on lista järjestelmän tekemistä käännösehdotuksista.

- 2.2. Perusteet käännösehdotuksen valintaan, arvosana
  Hyvyysarvo palautetuille käännöksille. Hyvyysarvo on suomenkielisten jäsennyspuiden etäisyys.
- 2.3. Kohdennetut virkkeet, hyvyysarvo
  Syöteasiakirjoissa olleet virkkeet kohdennettuina siten, että suomen- ja englanninkieliset vastinvirkkeet tai -virkeryhmät on löydetty. Jokaiselle parille annetaan lisäksi arvosana.
- 2.4. Tietokantavaste
  Näihin kyselyihin kuuluvat tietokantojen perustoiminnot kuten poisto-, listaus-, muokkaus- yms. pyynnöt.

4.2.2. Tietovirrat TranSmartin osana

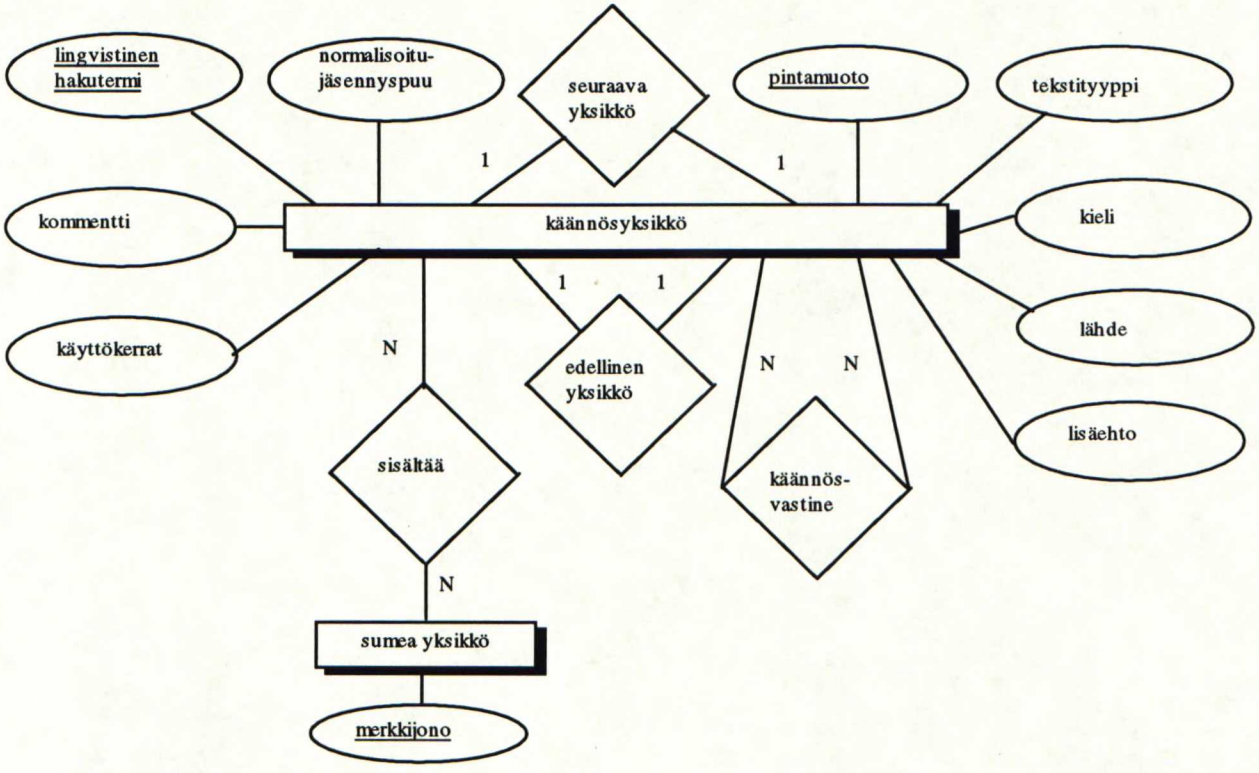
Konekäännösjärjestelmän syöte käännösmuistiin

- TranSmart-jäsennyspuu

Konekäännösjärjestelmän saama vaste käännösmuistista

- TranSmart-jäsennyspuu
  Syötteenä saadun puun käännös. Palautettavassa puussa kaikkien sanojen pitää olla oikein käännetty, vaikka tietokannasta ei olisikaan löydetty täysin yhteensopivaa käännösesimerkkiä. Jos sopivaa käännöstä ei saada aikaan, palautetaan siitä tieto.

4.3. Tietorakenteen kuvaus



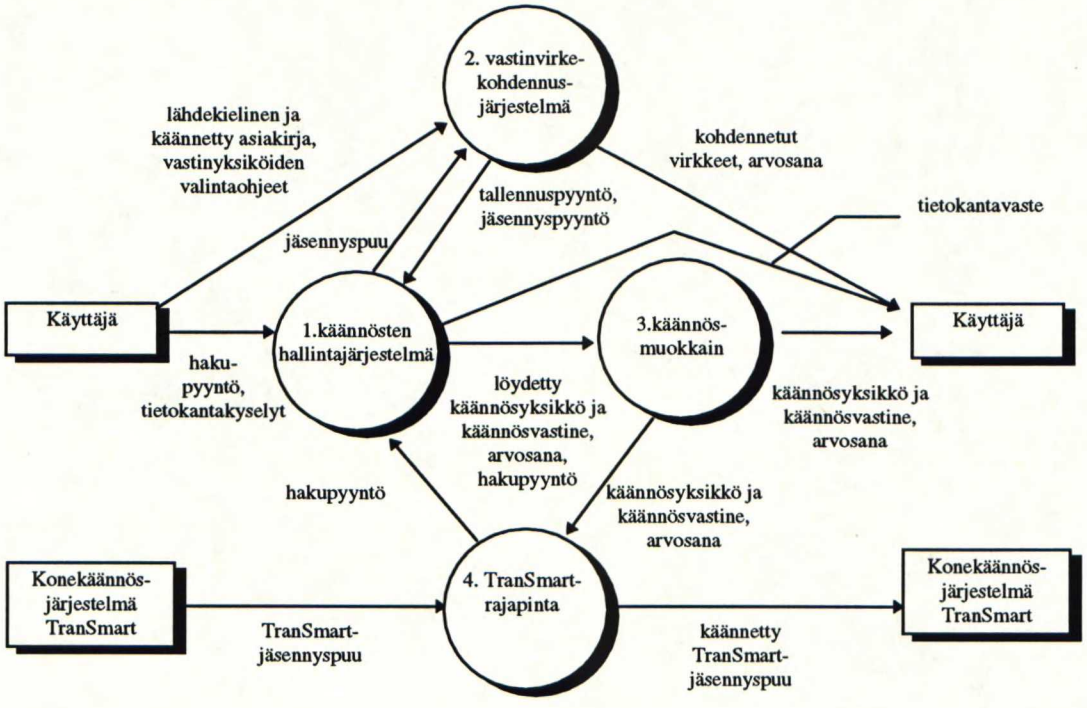
Kuva 4.2. ER-kaavio

Älykkääseen käännösmuistiin voi tallettaa jäsennettyjä käännösyksiköitä sekä niiden käännösvastineita. Käännösvastineet ja käännökset ovat kuitenkin tietokannassa samanlaisia olioita, koska molemmat ovat usein toistensa käännösvastineita ja käännösmuistia saatetaan haluta hyödyntää molempiin suuntiin. Käännösvastineita voi olla useita: eri kielisiä tai eri konteksteissa olleita.

Koska käännösmuistiin talletetaan edellinen ja seuraava käännösyksikkö, on mahdollista rekonstruoida alkuperäinen teksti. Näin käyttäjä pääsee tarkastelemaan myös sitä ympäristöä jossa virke on esiintynyt.

Tietokannassa tarvitaan myös olemassa olevista sumeista yksiköistä koostuva käänteisindeksi (inverted index), jonka avulla voidaan nopeuttaa hakuehdot osittain täyttävien virkkeiden hakua. Siihen tallennetaan esimerkiksi virkkeiden avainsanat, jolloin voidaan etsiä nopeasti kaikki virkkeet, joissa tietyt sanat esiintyvät.

4.4. Osajärjestelmät



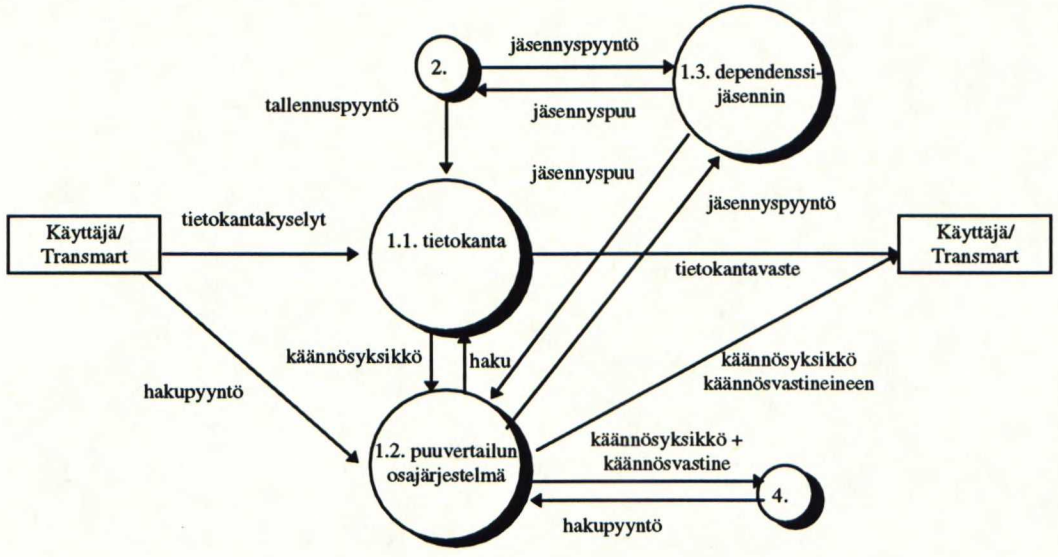
Kuva 4.3. (DFD taso 1) Osajärjestelmät

Varsinaisen **1. Ydinkäännösmuistin** lisäksi on eroteltu kolme muuta osajärjestelmää: **2. Vastinvirkkeiden kohdentamisjärjestelmä,** joka hakee vastinvirkkeet asiakirjoista, joiden tiedetään olevan toistensa käännöksiä. Kohdentamisjärjestelmää käytetään tietojen syöttämiseen käännösmuistiin; **3. Käännösmuokkain** muokkaa löydetystä, vain osittain käännettävänä olevan kanssa samanlaisen lähdekielisen virkkeen käännöksestä oikean käännöksen; **4. TranSmartin rajapintana** toimiva osajärjestelmä muuttaa jäsennyspuita TranSmart-järjestelmän ja käännösmuistin esitysmuotojen välillä. Sitä tarvitaan käännösehdotuksen liittämiseen käännösjärjestelmän muuhun toimintaan. Tarvittaessa tämä osajärjestelmä voi myös suorittaa käännettävän virkkeen osapuiden haun, joskin se tarvitsee siihen apua muilta osajärjestelmiltä.

5. Älykkään käännösmuistin toteutus

5.1. Käännösmuistin ydinosa

5.1.1. Ydinkäännösmuistin rakenne



Kuva 5.1. (DFD taso 2) Ydinkäännösmuistin osajärjestelmät

Käännösmuistin ydinosa suorittaa käännösmuistin perustehtävät. Se ylläpitää esimerkkikäännöksien tietokantaa ja mahdollistaa hakujen tekemisen. Haku tapahtuu yksikielisesti lähdekielisen virkkeen tai sen osan perusteella, kohdekielen ominaisuuksia ei käytetä. Puuvertailun osajärjestelmä (1.2) laskee depedenssijäsennyspuiden etäisyyden sekä luo sopivan hakutermin hajautusta varten. Jäsennyspuiden tuottamiseen tarvitaan virkkeiden depedenssijäsennin (1.3.). Varsinaiset tietokantatoiminnot (1.1) ovat kieliriippumattomia ja voidaan toteuttaa useimmilla valmiilla tietokantaohjelmilla. Luonnollisen kielen virkkeiden sumeaa hakua nopeuttavat ominaisuudet kuitenkin helpottavat toteutusta ja sallivat kattavampien hakutulosten saamisen.

Käännösmuistista voidaan hakea käännösyksiköitä pääasiassa kahdella tavalla: 1. tiettyä käännösyksikköä sen pintamuodon avulla tai 2. sumeasti käännösyksikköjoukkoa pintamuodon tai jäsennyspuun avulla. Jos käytettävissä on etsittävän käännösyksikön pintamuoto ja halutaan täydellinen yhteensopivuus (tapaus 1), on haku nopea ja toteutettavissa tehokkaasti useimmilla tietokannoilla niiden perusominaisuuksia käyttäen.

Suomea haku asettaa suurimmat vaatimukset tietokannalle. Periaatteessa voitaisiin käydä läpi koko tietokanta ja verrata kaikkia talletettuja jäsennyspuita etsittävään (exhaustive search). Käännösmuistin koko voi kuitenkin potentiaalisesti olla hyvinkin suuri. Aikaresurssien rajallisuuden takia joudutaankin hakua ohjaamaan ja hakuavaruutta karsimaan indeksoinnin avulla.

Ydinkäännösmuisti on siis eräänlainen tekstin haun sovellus. Sitä lähellä olevia ongelmia ovat mm. tiedonhaun (IR, Information Retrieval), vapaan tekstin haun (Full Text Search) tai luonnollisen kielen haun (Natural Language Search) ongelmat.

5.1.2. Menetelmiä tekstin haussa

5.1.2.1. Vapaan tekstin haun tehostaminen

Käänteisindeksi

Muun muassa tiedonhaun (IR) sovelluksissa on luonnollisen kielen sumeaa hakua varten kehitetty paljon erilaisia indeksointimenetelmiä. Eniten käytetty on **käänteisindeksi** (inverted index).

Käänteisindeksiä voidaan käyttää virkkeiden, asiakirjojen tms. haun nopeuttamiseen. Käänteisindeksi on luettelo sanoista ja niiden esiintymispaikoista. Jos kyseessä ovat suuret tekstiyksiköt, kuten kokonaiset asiakirjat, käänteisindeksi voidaan muodostaa avainsanoista. Virkkeiden tapauksessa voidaan käänteisindeksiin tallettaa jopa kaikki sanat.

Kun tietokannasta pitää hakea jokin virke, etsitään käänteisindeksistä virkkeen jokaista sanaa vastaava virkejoukko. Tähän käytetään nopeaa hakumenetelmää kuten hajautusalgoritmia. Löydetyistä virkejoukoista etsitään ne virkkeet, jotka kuuluvat mahdollisimman moneen joukkoon. Virkkeille annetaan arvosana sen mukaan, miten monessa joukossa ne esiintyvät. Mitä korkeamman arvosanan virke saa, sitä lähempänä se on etsittävää virkettä.

Tätä menetelmää käytetään mm. Internetin hakuohjelmassa WAIS (Wide Area Information Server). Paras tulos saavutetaan, jos käänteisindeksi tehdään sanojen perusmuodoista, koska tällöin päästään parempaan saantiin haun aikana. Koska älykkäässä käännösmuistissa on käytössä jäsennyspuu, voidaan käänteisindeksi tehdä siihen talletettujen sanojen perusmuotojen pohjalta. Näin käänteisindeksistä tulee mahdollisimman pieni, koska siinä on vain sanojen oikeat tulkinnat.

N-grammit

N-grammit ovat N:n kirjaimen pituisia merkkijonoja. Varsinkin englannin kielen tutkimuksen tuloksena on noussut ajatus **trigrammien** käytöstä: sanojen tai niiden perusmuotojen sijasta käänteisindeksi muodostetaan virkkeen trigrammeista eli sen kolmen peräkkäisen kirjaimen mittaisista yksiköistä. Esim. lauseen *Kissa kävelee* trigrammit olisivat:

kis, iss, ssa, sak, akä, käv, ävе, vel, ele ja lee

Myös virkkeen sanajärjestys vaikuttaa siitä löytyviin trigrammeihin. Sanajärjestyksen vaikutus ei kuitenkaan ole kovin suuri, jos sanat ovat pitkiä. Esimerkkivirkkeen kymmenestä trigrammista kaksi, *sak* ja *akä* aiheutuvat sanajärjestyksestä. Sanat eivät taivu kovin usein englannin kielessä, joten sanojen morfologinen analysointi voidaan jättää tekemättä. Suomenkielessa menetelmä ei liene yhtä käyttökelpoinen, ellei myös trigrammien muodostamiseen käytetä sanojen perusmuotoja (*kissa kävellä):*

kis, iss, ssa, sak, akä, käv, ävе, vel, ell ja *llä*

Tri- tai muita N-grammeja voidaan käyttää myös villikorttihakujen tehostamiseen. Jos esimerkiksi käytetään trigrammeja, voidaan tehokkaasi etsiä villikorttihakuja, kunhan kyselyssä on annettu ainakin kolme merkkiä (esim. **sak**). Tällöin ei voida käyttää sanojen perusmuotoja trigrammien muodostamiseen.

Kirjainten esiintymistiheys

Kirjainten esiintymistiheyksiäkin voidaan käyttää tekstinhaun tehostamiseen seuraavan algoritmin avulla:

Virkkeen tallettaminen:
  Tee virkkeestä histogrammi kirjainten esiintymistiheyksien mukaan

Virkkeen hakeminen :
  Anna etsittävää virkettä lähellä oleva virke
  Laske sen kirjainten esiintymistiheydet
  Etsi tietokannasta lähellä olevat kirjaintiheysvektorit

Tätä melko eksoottista menetelmää on käytetty lehtiartikkelien hakuun: muutamalla virkkeellä kuvataan etsittävä artikkeli. Näistä virkkeistä lasketaan kirjainten esiintymistiheydet ja etsitään sopiva lehtiartikkeli.

Käänteiset virkkeet

Käännösyksiköt voidaan myös järjestää sekä tavalliseen aakkosjärjestykseen, että kirjainjärjestykseltään käännettyjen virkkeiden mukaiseen aakkosjärjestykseen. Jos tietokannassa on mahdollista liikkua indeksin mukaisessa järjestyksessä, eli se ei perustu hajautukseen (hash), voidaan näin kahden indeksin avulla löytää virkkeet, joiden alku tai loppu ovat lähellä etsittävää.

Hajautus avainsanojen mukaan

Indeksointi voi perustua myös **avainsanoihin** ja **hajautusalgoritmiin**. Niiden avulla on mahdollista etsiä ne virkkeet, joissa on täsmälleen etsittävän virkkeen avainsanat. Avainsanojen järjestys voi vaihdella.

Ehtona algoritmin soveltamiselle on se, että virkkeiden avainsanat löydetään. Jokainen avainsana hajautetaan erikseen merkkijonojen hajautukseen sopivalla algoritmilla. Avainsanojen hajautusavaimet (hash values) lasketaan yhteen tai sovelletaan niihin jotain muuta symmetristä, assosiatiivista operaattoria. Tarpeen mukaan voidaan soveltaa toista hajautusta. Näin voidaan hyvin tehokkaasti etsiä virkkeet, joissa halutut avainsanat ovat. Tämän toteuttamiseen tarvitaan tietokanta, jossa voi määritellä omia hajautusfunktioita [comp.ai.nat-lang, 1995]

Piirrevektorit

Jos käytössä on tehokas luonnollisen kielen jäsennin tai sananmuotojen merkitsijä (tagger), voidaan avainsanojen hajautusideaa viedä pidemmälle ja muodostaa piirrevektoreita, joita käytetään haussa. Näin voidaan hakuun liittää myös semanttisia piirteitä esim. tesauruksien avulla. [Sutcliffe et al., 1995]. Tällainen piirrevektori jakaa aineiston ekvivalenssiluokkiin, joiden sisällä olevat virkkeet katsotaan toisiaan lähellä oleviksi. Eri luokkiin kuuluvien virkkeiden etäisyys tulkitaan suureksi. Käyttämällä piirrevektoria haun indeksointiin saadaan erittäin nopea haku, koska yksi haku riittää koko sumean joukon noutamiseen. Toisaalta piirrevektori pitää määritellä etukäteen, joten menetelmä ei ole kovin joustava.

Muita menetelmiä

Edellä esitettyjen menetelmien lisäksi lienee olemassa erilaisia kokeiluja muilla menetelmillä. Esimerkiksi esiintymistiheyksiä voidaan laskea sanaluokkien, termien tms. yksiköiden mukaan. Hakutermeinä voidaan käyttää myös esimerkiksi sanan semanttista luokkaa. Todellinen käyttötarkoitus ja käytössä olevat luonnollisen kielen analysointityökalut vaikuttavat hakumenetelmän valintaan.

Älykkäässä käännösmuistissa ei välttämättä tarvita perinteistä käänteisindeksiä sumeaan hakuun, koska jäsennyspuista on mahdollista muodostaa mielekäs yksikäsitteinen lingvistinen piirrevektori sumeaa hakua ohjaamaan. Käänteisindeksiä tarvitaan kuitenkin, jos halutaan hakea käännösyksiköitä, joita ei osata jäsentää tai jos halutaan etsiä tietokannasta käännösyksikköjen osia. Käyttäjä voi esimerkiksi haluta etsiä niiden virkkeiden käännökset, joissa esiintyy termi *Maailman Pankki*.

5.1.2.2. Käännösyksiköiden vertailun menetelmiä

Kun on löydetty joukko käännösyksiköitä, joiden tiedetään heuristisesti olevan suhteellisen lähellä etsittävää, jää ongelmaksi oikean käännösyksikön valitseminen. Tavoitteena on löytää sellainen käännösyksikkö, jonka kohdekielinen vastine olisi mahdollisimman lähellä käännettävän tekstin oikeaa käännöstä. Nykyisissä kaupallisissa käännösmuisteissa vertailu perustuu pääasiassa tekstin pintamuotoon.

Toisaalta esimerkkiperusteisissa käännösjärjestelmissä käännösyksiköiden vertailua on jouduttu tutkimaan enemmän, koska tarkkuusvaatimuksen ovat olleet suuremmat. EBMT-tutkimuksen tuloksista löytyykin joitain kielellisesti perusteltuja menetelmiä tekstin vertailuun. Esimerkiksi tesaurukset ja lauseiden syntaktinen tieto tarjoavat sopivaa lisäinformaatiota vertailun pohjaksi.

Pintavirkkeiden merkkijonovertailu

Yksinkertaisin ja kieliriippumattomin tapa on vertailla käännösyksiköiden lähdekielisiä **pintamuotoja** ja vaatia täydellinen yhteensopivuus. Jos käännettävä virke on sama kuin jokin tietokannasta löytyvä, käytetään tämän käännöstä. Muussa tapauksessa kääntäminen jätetään käyttäjän tehtäväksi [Nirenburg et al., 1993]. Tämä saattaa kuulostaa ongelman yksinkertaistamiselta, mutta menetelmää käytetään ja tarvitaan käännösteollisuudessa ainakin asiakirjojen uusia versioita käännettäessä (ns. revision problem). On olemassa kääntäjiä, jotka haluavat vain oikean käännöksen, jos sellainen on olemassa. Heille ei siis kelpaa osittain oikea käännös, vaikka siitä hieman muokkaamalla saa oikean käännöksen.

Käännösyksiköiden pintamuotojen vertailua voidaan sumentaa. **Sumea merkkijonovertailu** perustuu tekstin pintamuotoon. Jotta löydettäisiin lähin talletettu käännösyksikkö, lasketaan käännösyksiköiden samanlaiseksi tekemiseen tarvittavat tekstinkäsittelykomennot. Näiden painotetusta lukumäärästä saadaan lukuarvo merkkijonojen etäisyydelle (ns. edit distance). Sumeaa merkkijonovertailua varten löytyy useita algoritmeja [Jokinen et al., 1991].

Kielellisten piirteiden käyttö vertailussa

Merkkijonoperusteinen vertailu toimii hyvin, jos vaaditaan aina, että pintavirkkeet ovat suhteellisen lähellä toisiaan. Tyypillistä ihmisten käyttämälle kielelle on kuitenkin se, että jokin asia voidaan ilmaista monella eri tavalla. Merkkijonovertailuun perustuva käännösmuisti ei löydä sisällöltään samanlaisia, mutta ulkoasultaan poikkeavia virkkeitä. Ideaalinen käännösmuisti perustuisikin semanttiseen analyysiin. Semanttinen analysointi on vielä turhan raskas menetelmä tällaiseen sovellukseen. Vertailua voidaan kuitenkin viedä siihen suuntaan erilaisten heuristiikkojen avulla.

Merkkijonoperusteisen vertailun yhteydessä ei myöskään voida tehdä kovin paljon muutoksia kohdekieliseen virkkeeseen, jos etsittävä ja tietokannassa oleva virke eivät ole täysin samoja. Älykkään käännösmuistin ominaisuuksiin kuitenkin kuuluu myös kyky muokata kohdekielisiä virkkeitä lähemmäksi oikeaa käännöstä. Tämän edellytyksenä on virkkeen korjattavissa olevien ominaisuuksien tunnistaminen. Ne eivät näy suoraan virkkeen pintamuodosta.

Merkkijonovertailun tarkkuutta voidaan parantaa muunmuassa sanojen **morfologisen tiedon** avulla eli esimerkiksi tiedolla sanojen perusmuodoista tai monikon päätteistä. Käsiteltävien virkkeiden sanoista voidaan näin karsia vähemmän oleellisia piirteitä tai vaikkapa muuttaa ne perusmuotoon.

Jeremy Caroll on kehittänyt **samankaltaisuuskulman** käsitteen (angle of similarity). Hän käyttää menetelmässään kolmiota, jonka kaksi pistettä edustavat vertailtavia virkkeitä ja kolmas nollavirkettä. Järjestelmä sisältää joukon ad hoc -sääntöjä, jotka määrittelevät ne tuhoamis-, lisäämis- ja muunto-operaatiot, jotka tarvitaan toisen virkkeen muokkaamiseen toisen kaltaiseksi. Näiden operaatioiden painojen avulla saadaan kolmion sivujen pituudet. Caroll esittää nollavirkettä vastaavassa pisteessä olevan kulman laskemiseen perustuvan määritelmän virkkeiden etäisyyden laskemiseen [McLean, 1992]. Kulman käsite on laskennallinen apuväline etäisyyden laskemiseen; lingvistinen tieto on ad hoc -säännöissä, joita McLean (1992) ei kuvaa.

McLean (1992) esittää etäisyysongelman ratkaisuksi **neuraaliverkkoa**. Neuraaliverkot soveltuvat hyvin esimerkkiperusteiseen kääntämiseen ja sumeaan hakuun. Niillä olisi luontevaa yhdistää virkkeiden erilaiset kielelliset ominaisuudet. Hänen järjestelmänsä on kuitenkin varsin yksinkertainen ja testaakin vain neuraaliverkkojen soveltuvuutta. Se mm. perustuu sanojen absoluuttiseen sijaintiin virkkeiden pintamuodossa, eikä esimerkiksi huomaa lainkaan seuraavien virkkeiden samankaltaisuutta:

Jaakko on puussa keräämässä omenia.
Virtasen Jaakko on keräämässä omenia puussa.

Käytännön toteutus vaatisi huomattavasti suuremman määrän neuroneita.

D.B. Jones (1991) kuvaa väitöskirjassaan **piirrevektoreihin** perustuvan etäisyyden laskentamenetelmän. Esimerkkikäännöksiin voidaan liittää useita vektoreita, jotka kuvaavat niiden eri tasoisia piirteitä, esimerkiksi morfologisia ominaisuuksia. [McLean, 1992].

Sato ja Nagao (1990) ottavat käännösyksiköiden vertailemiseen mukaan myös niiden **ympäristön**. Heidän järjestelmänsä on esimerkkiperusteinen käännösjärjestelmä. He etsivät esimerkkivirkkeistä osia, jotka soveltuvat käännettävän virkkeen osien käännöksiksi. Sopivia osia löytyy yleensä melko helposti, koska palasia voidaan jakaa pienempiin osiin kunnes sopivat käännökset löytyvät. Niinpä valintakriteereiksi nousevat:

1. käännösyksikön pituus: pidempi käännösyksikkö on lyhyempää parempi,

2. ympäristöjen etäisyys: mitä lähempänä toisiaan käännettävän tekstisegmentin ja esimerkkikäännöksen ympäristöt ovat, sitä parempi esimerkkikäännös on.

Sanojen etäisyyksiä käytetään käännösyksikön **rajoitettujen ympäristöjen** (restricted environment) etäisyyksien laskemiseen. Rajoitettuun ympäristöön kuuluvat alipuun naapurisolmut. Näistä eri tekijöistä lasketaan käännösyksikön arvosana (score).

Semanttinen vertailu tesauruksien avulla

Tesauruksia eli sanastoja, joissa sanat on luokiteltu semanttiseen hierarkiaan, voidaan käyttää paitsi yksittäisten sanojen semanttisen etäisyyden määrittelemiseen, myös virkkeiden etäisyyksien laskemisen apuvälineenä.

Sumitalla, Iidalla ja Kohyamalla on käytössään suuri tesaurus. He määrittelevät kahden sanan etäisyyden sen mukaan, miten kaukana on niiden yhteinen esi-isä semanttisessa hierarkiassa.

Näin saatu paino yhdistetään tilastollisiin menetelmiin perustuvaan sanan sanaluokkatietoon. [McLean, 1992].

Furusen ja Iidan (1992) tesauruksessa sanoihin on liitetty joukko semanttisia attribuutteja, joita käytetään sanan sijaintipaikan lisäksi sanojen etäisyyden määräämiseen. Päinvastoin kuin tekstinhaussa yleensä, heidän ongelmanaan on oikean käännöksen valitseminen monien mahdollisuuksien joukosta. Niinpä Furuse ja Iida laskevat etäisyyttä kohdekielisten ja lähdekielisten tekstisegmenttien välillä. Virkkeiden etäisyydet saadaan sanojen etäisyyksien painotettuna summana. Painoina käytetään sanojen sijaintipaikan painoa.

Yleisiä sääntöjä etäisyysmetriikan laatimiseen

Nirenburg, Domashnev ja Grannes (1993) esittävät **yleiset säännöt** käännösyksiköiden etäisyyksien laskemisen perustaksi:

1. määrittele joukko vertailukriteerejä yksittäisille sanoille,
2. määrittele paino vertailukriteerien rikkomiselle,
3. määrittele kaava, jolla yksittäisten sanojen painoista saadaan koko tekstisegmentin etäisyys,
4. tarkista saadut parametrit testisaineistolla, jossa on erikseen määritelty, millaisten hakujen tuloksena tiettyjä virkkeitä halutaan saada aikaan.

Näiden sääntöjen pohjalta he ovat kehittäneet melko kattavan joukon yksittäisten sanojen vertailukriteerejä. He vertaavat sanojen **pintamuotoja**, niiden **morfologisia ominaisuuksia**, niiden tesauruksesta löytyviä **synonyymejä, hyponyymejä** jne. Tätä vertailukriteerijoukkoa käyttäen he ovat laatineet testituloksien avulla viritetyn kaavan kahden lähdekielisen tekstisegmentin S:n ja S':n etäisyyden laskemiseen:

$$S - S' = 20W + 10w + 5H + 4Y + 3M + 0C \quad (= \text{etäisyys})$$

S = käännettävä tekstisegmentti
S' = talletettu esimerkkikäännös
W = niiden sanojen lkm, jotka ovat S:ssä, mutta eivät S':ssa
w = niiden sanojen lkm, jotka ovat S':ssä, mutta eivät S:ssa
H = niiden sanojen lkm, joille löytyy vain yhteinen hyper-tai hyponyymi
Y = niiden sanojen lkm, joille löytyy vain yhteinen synonyymi
M = niiden sanojen lkm, joille löytyy yhteys vain morfologisen analyysin perusteella
C = niiden sanojen lkm, jotka ovat täysin samanlaisia

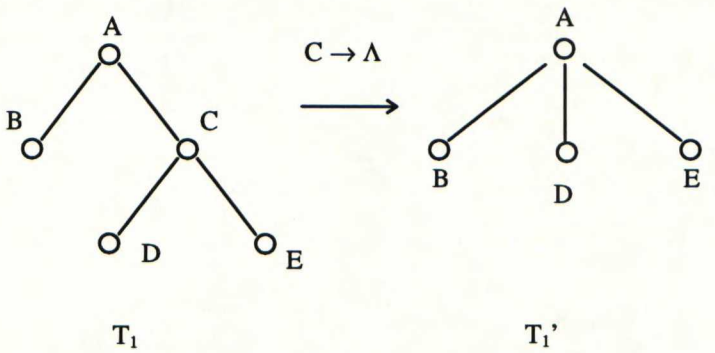
5.1.3. Puiden etäisyyden laskemisen teoriaa

Puiden etäisyyden määritelmä

Kun käytössä on virkkeiden jäsennyspuut ja halutaan laskea niiden etäisyys mahdollisimman tarkasti, päädytään yleisten puiden painotetun etäisyyden laskemisongelmaan. Ottamalla vertailussa huomioon puiden topologia, solmujen väliset relaatiot ja niiden attribuutit, käytetään semantiikkaa lukuunottamatta kaikki saatavilla oleva tieto virkkeiden vertailemiseen. Olemassa olevissa kaupallisissa tietokonelingvistisissä järjestelmissä ei tiettävästi ole toteutettu jäsennyspuiden topologiaan perustuvaa käännösyksiköiden etäisyysmittausta.

Puiden etäisyys määritellään tarvittavien **puumuunnosoperaatioiden kustannuksien** minimisummana. Puumuunnosoperaatioilla simuloidaan sitä operaatioketjua, joka tarvitaan vertailtavien puiden muuttamiseksi samanlaisiksi.

Muunnosoperaatio kohdistuu aina johonkin solmuun. Muunnosoperaatiota merkitään c→b:llä, missä c on korvattava solmu ja b uusi solmu. Λ:llä merkitään tyhjää solmua. On olemassa kolme erityyppistä puumuunnosoperaatiota: 1. solmun poistaminen (c → Λ), 2. solmun lisääminen (Λ→ c), ja 3. solmun muuntaminen (c → b). Jokaiseen solmuun juurta lukuunottamatta liittyy yksikäsitteinen, solmun isäänsä kytkevä kaari, joka poistetaan tai lisätään solmun mukana. [Tai, 1979]. Esimerkiksi solmun C poistamisella kuvan 5.2 T_1:stä on seuraava vaikutus:



Kuva 5.2. Solmun poistaminen

Painofunktio φ() on positiivinen reaaliluku, joka määrittää muunnosoperaation c→b kustannuksen φ(c→b). Koska φ() kuvaa etäisyyttä, ovat seuraavat ehdot tosia:

$\varphi(a \rightarrow b) \geq 0$,
$\varphi(a \rightarrow a) = 0$,
$\varphi(a \rightarrow b) = \varphi(b \rightarrow a)$ ja
$\varphi(a \rightarrow b) + \varphi(b \rightarrow c) \geq \varphi(a \rightarrow c)$.

Jos S on sarja muunnosoperaatioita $S=(a_1 \rightarrow b_1, a_2 \rightarrow b_2, .. , a_n \rightarrow b_n)$, voidaan painofunktio φ() määritellä S:lle sen muunnosoperaatioiden kustannusten summana:

$$\varphi(S) = \sum \varphi(a_i \rightarrow b_i)$$

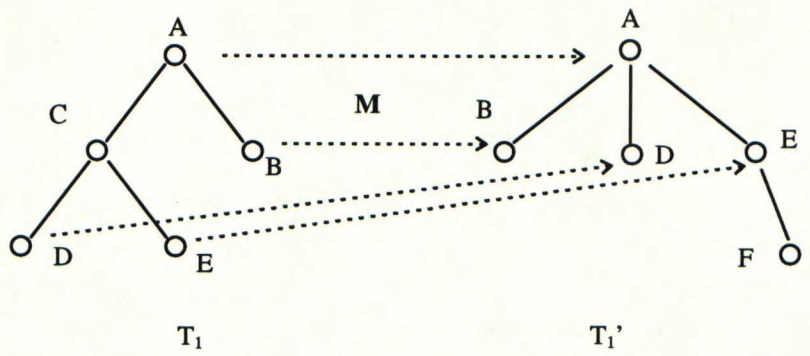
Kahden puun T_1:n ja T_2:n **etäisyys** $\delta(T_1, T_2)$ määritellään seuraavasti:

$$\delta(T_1, T_2) = \min_S \{\varphi(S) \mid S \text{ on muunnosoperaatioiden sarja, joka muuttaa } T_1\text{:n } T_2\text{:ksi}\}$$

Myös δ() kuvaa etäisyyttä φ():n määritelmän seurauksena.[Shasha et al., 1994]

Puun kuvautuminen toiselle

Muunnosoperaatioiden avulla on mahdollista määritellä graafisesti mitä tarkoittaa puun **kuvautuminen** (mapping) toiselle. Kuvaus M on sarja muunnosoperaatioita, jotka sovelletaan toiseen puuhun. [Zhang et al., 1989]



Kuva 5.3. Puun kuvaus M

Puiden kuvaukset ovat havainnollinen tapa esittää puiden erot ja yhtäläisyydet.

Puiden etäisyyksien laskeminen

Kun puumuunnosoperaatioille määritellään attribuuteista riippuvat painot, löytyy kahden **järjestetyn** puun etäisyyden laskeva algoritmi, joka on aikavaativuudeltaan

$$O(|T_1| * |T_2| * \min(\text{depth}(T_1), \text{leaves}(T_1)) * \min(\text{depth}(T_1), \text{leaves}(T_1)))$$

ja tilavaativuudeltaan

$$O(|T_1| * |T_2|)$$

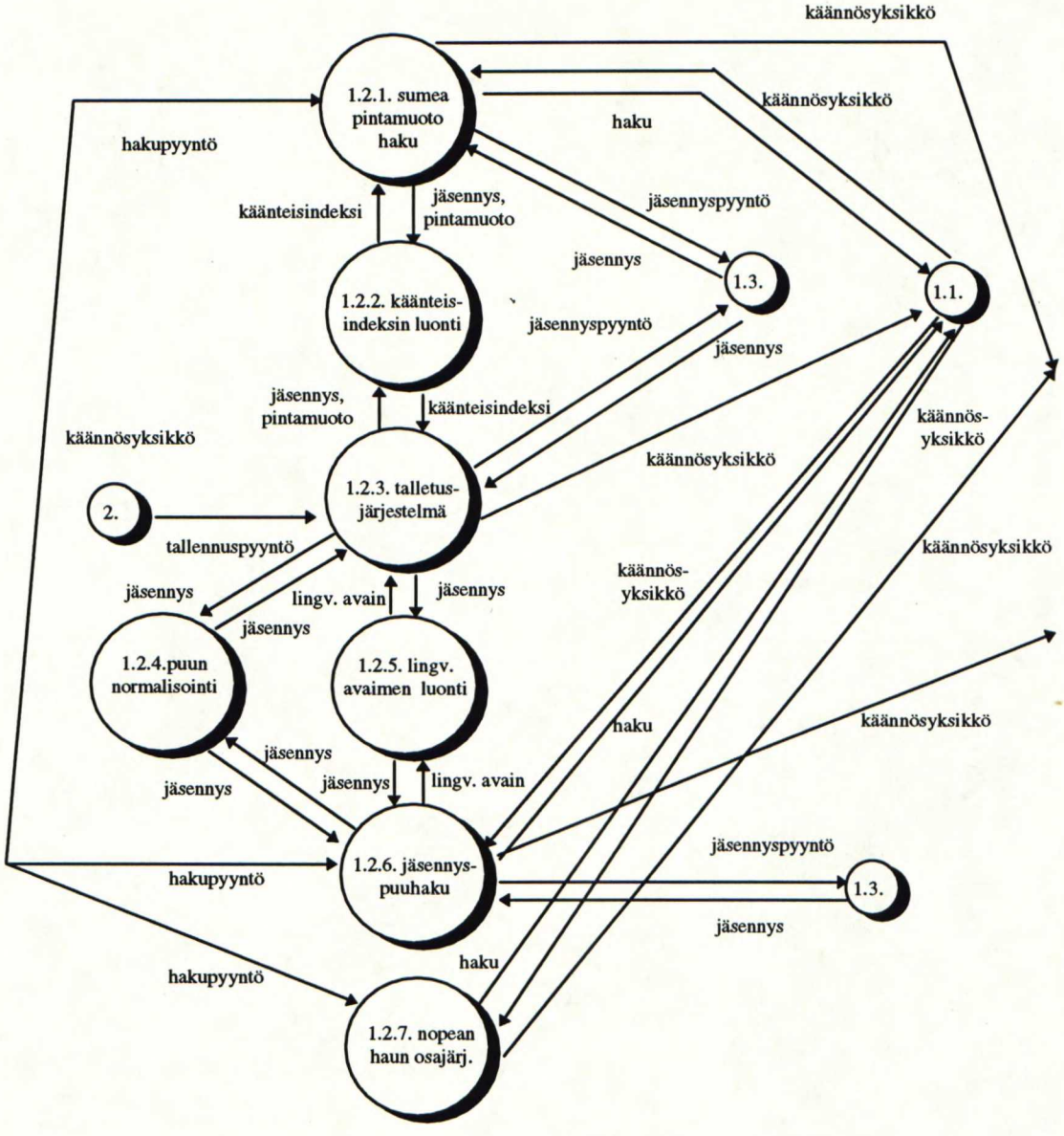
[Zhang et al., 1989].

Luonnollisen kielen depedenssijäsennyspuut ovat **järjestämättömiä** puita: tietyn jäsennyspuun jonkin solmun välittömät lapset voidaan esittää kaikissa mahdollisissa järjestyksissä ilman, että puun merkitys muuttuu. Järjestämättömien puiden etäisyyden laskeminen on NP-täydellinen ongelma. Jos kummankaan puun koko ei ole rajoitettu, joudutaan käyttämään aikavaativuudeltaan **eksponentiaalista** algoritmia puiden etäisyyden laskemiseen [Shasha et al., 1994]. Luonnollisen kielen virkkeille ei löydy teoreettista maksimipituutta, joten myöskään niiden jäsennyspuiden koko ei ole rajoitettu.

Järjestämättömien puiden etäisyyden laskemisen aikavaativuutta on mahdollista parantaa yleisten heuristiikkojen avulla. Heuristiikat voivat perustua paikallisen minimin hakemiseen, ns. simuloitu jäähdytys -tekniikkaan (simulated annealing), järjestämättömien puiden järjestämiseen tms. [Shasha et al., 1994]. Heuristiikoista huolimatta yleiset puiden vertailualgoritmit ovat käytännössä hitaita älykkääseen käännösmuistiin. Depedenssijäsennyspuiden kielellisiin ominaisuuksiin nojautuen vertailua voidaan kuitenkin edelleen tehostaa.

5.1.4. Ydinkäännösmuistin toteutus

5.1.4.1. Puuvertailun osajärjestelmän rakenne ja toimintaperiaate



Kuva 5.4. (DFD taso 3) Puuvertailun osajärjestelmä

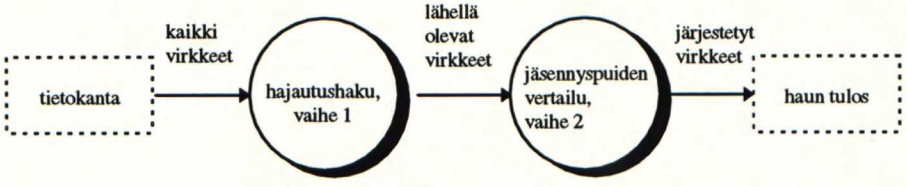
Puuvertailun osajärjestelmän vastuulla on depedenssijäsennyspuiden käsittely: niiden etäisyyksien laskeminen, normalisointi ja hakutermien muodostaminen.

Ennen kuin virke voidaan tallettaa käännösmuistiin, se pitää jäsentää. Jäsennyspuu normalisoidaan TranSmartin Oksaimen ja erillisen käännösmuistin normalisointisäännöstön avulla. Puusta muodostetaan hakutermi sumeaa hakua varten. Näin saatu tietorakenne talletetaan tietokantaan.

Hakupyyntö käännösmuistiin on lähdekielisen tekstisegmentin pintamuoto eli yleensä yksi virke tai valmiiksi jäsennetyn virkkeen jäsennyspuu. Jos hakupyyntö on jäsentämätön virke, se

jäsennetään TranSmartin jäsentimellä. Jäsennyspuu normalisoidaan Oksaimella ja erillisellä säännöstöllä, kuten on tehty kaikille talletetuillekin virkkeille.

Haku jaetaan kahteen vaiheeseen tehokkuussyistä. Ensin tietokannasta haetaan älykkään hakutermin avulla joukko käännösyksiköitä. Näiden tarkempi etäisyys käännettävään tekstiin lasketaan haun toisessa vaiheessa, jossa myös poimitaan kielellisten ominaisuuksien perusteella parhaiten soveltuvat käännökset (kuva 5.5).



Kuva 5.5. 2-vaiheinen hakuprosessi

Älykkäässä käännösmuistissa hakutermien muodostamiseen käytetään suomenkielisten virkkeiden normalisoituja jäsennyspuita. Niiden avulla voidaan luoda lingvistisesti mielekkäitä hakutermejä. Jäsennyspuun topologiaan perustuen voidaan hakutermi muodostaa puun juuresta ja tarvittaessa sen välittömistä jälkeläisistä. Termin tulisi olla sellainen, että sen avulla löydetään varmasti kaikki sumean haun edellyttämät käännösyksiköt. Niille kielille, joille ei ole valmista depedenssijäsennintä, joudutaan käyttämään käänteisindeksiä. Sitä tarvitaan myös muun muassa virkkeiden sisällä olevien fraasien etsimistä varten.

Haun toisessa vaiheessa etsitään se käännösyksikkö, jonka etäisyys etsittävään on pienin. Haku on parametroitavissa erillisellä kielellisellä normalisointisäännöstöllä. Sillä normalisoidaan samanlaisiksi sellaiset kielelliset ilmaisut, joiden välillä ei haluta tehdä eroa vertailun aikana. Normalisointi tehdään ennen virkkeiden tallentamista. Näin saavutetaan nopea algoritmi, koska kielellistä tietämystä ei tarvitse soveltaa haun aikana kuin etsittävään käännösyksikköön. Heikkoutena on tarve käännösmuistin virkistämiseen jäsentimen ja normalisointisääntöjen päivitysten yhteydessä.

5.1.4.2. Käännösyksiköiden käsittelyvaiheet

Tekstin muotoilujen käsittely

Tekstin muotoilut eivät pääsääntöisesti vaikuta sen käännökseen kielellisesti. Niinpä muotoiluja ei käännösmuistissakaan yleensä tarvita. Kuitenkin tietyissä tapauksissa, joissa halutaan käännösmuistista täsmälleen oikea käännös, pitää myös muotoilut ottaa huomioon etsittäessä sopivaa käännöstä. Tätä varten muotoilut tallennetaan käännösmuistiin virkkeen pintamuodon mukana.

Muotoilujen tulee olla SGML-standardin mukaisina merkintöinä tekstin seassa. SGML on paljon käytetty, siirrettävä ja sopii hyvin luonnollisen kielen talletusmuodoksi. Se on myös TranSmart-järjestelmän käyttämä tiedon esitystapa.

Myös välimerkit tulee yhdenmukaistaa ennen virkkeiden tallettamista tai hakua. Kaikki rivinvaihdot tai suuremmat välit muutetaan yhdeksi välilyönniksi.

Virkkeiden jäsentäminen

Virkkeiden jäsentämiseen käytetään TranSmartin suomen kielen depedenssijäsennintä ja virkkeentunnistinta. Virkkeentunnistin muuttaa pintavirkkeen sanaketjuksi jäsennintä varten. Jäsennin tekee sanaketjusta depedenssikieliopin mukaisen jäsennyspuun tai -metsän, jos sanaketju ei ole yhtenäinen virke. Depedenssijäsennin neutraloi virkkeen lauseenjäsenten järjestyksen.

Jäsennyspuu talletetaan staattisiin muuttujiin ja sitä voidaan käsitellä kirjastofunktioiden avulla. TranSmart mahdollistaa vain yhden jäsennyspuun käsittelyn kerrallaan.

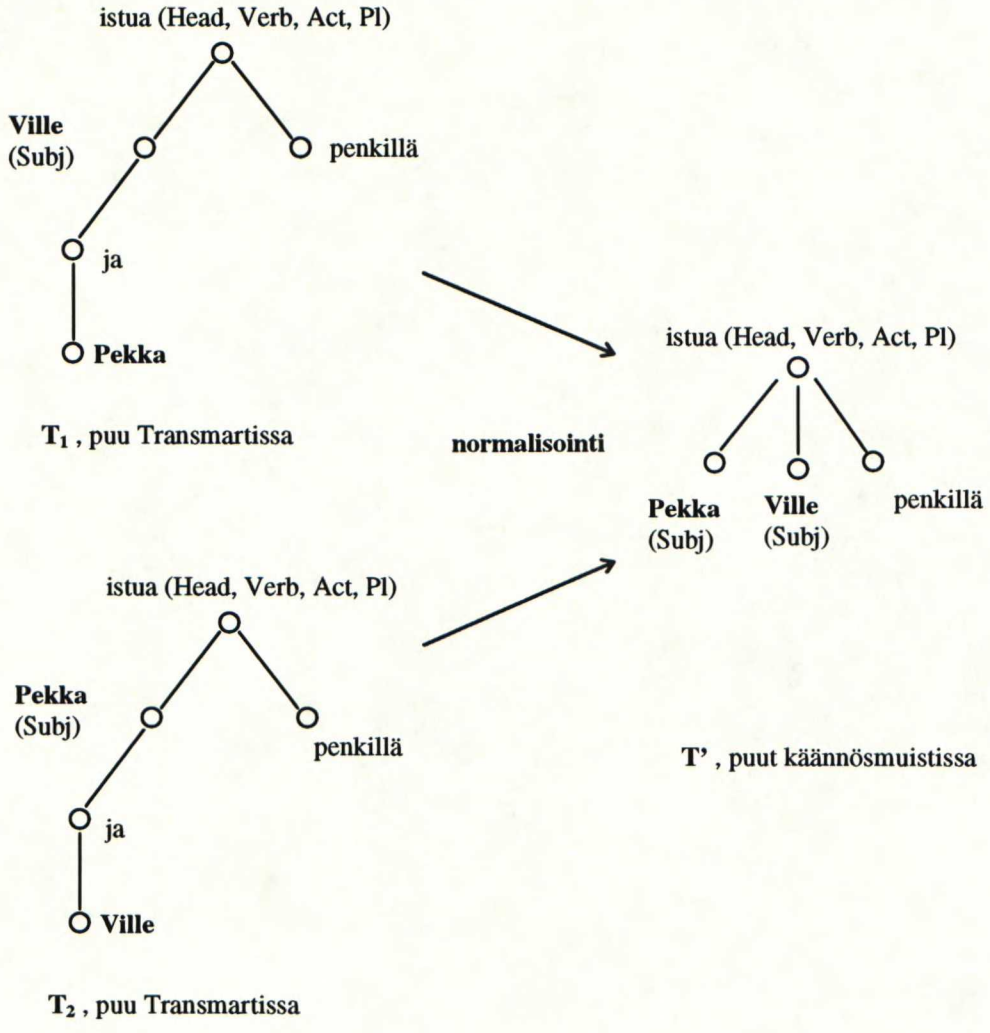
Jäsennyspuiden normalisointi

Jäsennyspuiden normalisoinnilla nopeutetaan niiden sumeaa vertailua ja mahdollistetaan haun parametrisointi. Normalisoinnin ensimmäinen vaihe tehdään käännösjärjestelmän Oksaimella, joka karsii puuta lingvistisesti perusteltujen sääntöjen avulla. Oksain muuttaa eräät puhtaasti syntaktiset ns. funktiosanat piirteiksi. Myös osa puun redundantista tiedosta poistetaan. Esimerkiksi virkkeen *Me rakennamme kesämökkiä* persoonapronomini *Me* karsitaan pois Oksaimessa merkityksen kannalta ylimääräisenä sanana.

Oksaimen jälkeen voidaan tehdä lisänormalisointia ja puun karsimista, jolla voidaan parametrisoida käännösmuistin haun sumeutta. Normalisoinnin tarve riippuu käyttökohteesta. Jos käännösmuisti toimii eräajokäytössä käännösjärjestelmän osana, ei puuta voida kovin paljon karsia, koska eräajossa ei ole ihmiskääntäjää valvomassa tehtyjen valintojen hyvyyttä. Tällöin käännösmuistista voidaan valita vain hyvin lähellä oikeaa käännöstä olevia käännöksiä ja puun kaikki oksat tulee ottaa mukaan vertailuun.

Toisaalta myös eräajokäytössä käännösmuistin normalisointisäännöillä voidaan muokata semantiikaltaan identtisiä puita toistensa näköisiksi, mikä helpottaa samankaltaisuuden huomaamista. Esimerkiksi solmun rinnastetut jälkeläiset ovat TranSmartin Jäsentimen ja Oksaimen jäljiltä ketjussa (T_1 ja T_2 kuvassa 5.6). Rinnasteiset solmut tulee nostaa isänsä välittömiksi lapsiksi (T' kuvassa 5.6), jotta puita vertailtaessa löydetään esimerkiksi seuraavien virkkeiden samankaltaisuus:

Pekka ja Ville istuvat penkillä.
Ville ja Pekka istuvat penkillä.

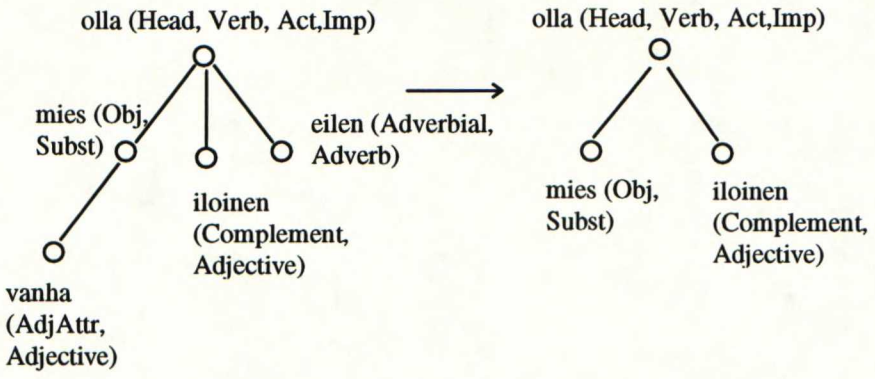


Kuva 5.6. Eräs mahdollinen tapa normalisoida jäsennyspuita

Näin voidaan järjestää mm. se, että varsinaisen puiden vertailualgoritmin ei tarvitse nostaa solmuja tasolta toiselle. Tämä nopeuttaa algoritmia huomattavasti.

Jos järjestelmä parametrisoidaan **interaktiiviseen** käyttöön, voidaan haun sumeutta lisätä. Normalisoinnin tavoitteena voi tällöin olla talletettavan käännösyksikön ytimen hakeminen eli sen syntaktisen perusrakenteen tunnistaminen. Talletettaville lauseille tämä on lauseen **verbikonfiguraatio** eli predikoiva verbi ja sen lauseenjäsenet [Arnola, 1995a].

Esimerkiksi virkkeestä *Vanha mies oli eilen iloinen* voidaan karsia adjektiiviattribuutti *vanha* ja adverbiaali *eilen*. Normalisoituna siitä saadaan *Mies oli iloinen*. Tällaista parametrisointia voidaan käyttää, jos tiedetään, että käännösehdotus tulee ihmiskääntäjän käyttöön (kuva 5.7).



Kuva 5.7. Jäsennyspuun normalisointi interaktiivisessa käännösmuistissa

Kun vertailu suoritetaan tämän karsitun rakenteen pohjalta, käyttäjä saa kääntäessään todennäköisemmin edes jonkin käännösehdotuksen, josta hän voi muokata mieleisensä. Tämän esimerkkivirkkeen käännöstä voidaan ehdottaa esimerkiksi virkkeiden *Tumma mies oli iloinen*, *Mies oli iloinen voitostaan* ja ehkä jopa *Ville oli surullinen* käännökseksi, ellei parempaa ehdotusta ole talletettu. Näin saadaan käännösmuistiin talletetusta tiedosta mahdollisimman suuri hyöty ja käyttäjä saa mielekkään käännöspohjan, jonka hän tai käännösmuokkain (kappale 5.6) voi korjata oikeaksi.

Normalisointi ei ole yhtä aikakriittinen operaatio kuin puiden varsinainen vertailu, koska normalisointi tehdään vain kerran etsittäessä ja kerran talletettaessa yhtä käännösyksikköä. Muun muassa tästä syystä normalisointi kannattaa tehdä TranSmartin dependenssipuiden manipulointiin tarkoitetulla sääntökielellä.

Puun solmujen normalisointi synonyymisanaston avulla

Eräänä ongelmana normalisoitujenkin puiden vertailussa ovat semantiikaltaan toisiaan lähellä olevat ilmaukset, joiden juuret kuitenkin ovat erilaiset. Juurelle nimittäin tulee välttämättä melko suuri paino puiden etäisyyksiä laskettaessa. Depedenssiteorian mukaan täydellisen virkkeen puun juuri on päälauseen finiitti verbi. Esimerkiksi:

Me korjasimme kesämökkiämme keväällä. (juurena *korjata*)
Me ehostimme kesämökkiämme keväällä. (juurena *ehostaa*)

Esimerkkivirkkeiden merkitykset ovat melkein samat ja molemmat voitaisiin kääntää englanniksi 'repair'-verbillä. Jos älykäs käännösmuisti käyttää haun ensimmäisessä vaiheessa hajautukseen puun juuren lekseemiä, saattaisi käydä niin, että näiden virkkeiden samanlaisuus jäisi kokonaan huomaamatta.

Muun muassa tämän ongelman poistamiseen ja semanttisesti toisiaan muistuttavien sanojen yhdenmukaistamiseen käytetään synonyymisanastoa, jossa keskenään samaa tarkoittavista sanoista on yksi valittu perustulkinnaksi.

Vielä parempi ratkaisu olisi täydellinen thesaurus, jossa sanat olisi luokiteltu semanttiseen hierarkiaan. Sen avulla hakutuloksista saataisiin semanttisesti perustellumpia. Tällaista laajaa thesaurusta ei kuitenkaan ole saatavilla suomenkielelle.

Älykkään hakutermin muodostaminen

Kun puu on normalisoitu, siitä voidaan muodostaa älykäs hakutermi. Se on tietokannassa sumean haun ensivaihetta varten (kuva 5.5).

Normalisoidun alipuun **juuri** on yleensä ilmaisun pääsana (esim. *istua* kuvassa 5.6). Lauseen normalisoidun jäsennyspuun juuri on sen predikoiva **verbi**. Hakutermi saadaan normalisoidusta jäsennyspuusta heuristisesti seuraavasti: juuresta talletetaan sen synonyymisanastosta haetun perustulkinnan perusmuoto. Tämän lisäksi avaimeen liitetään verbin välittömien jälkeläisten relaatioiden tunnisteet. Puussa esiintyviä adjektiiviattribuutteja tai adverbeja ei kuitenkaan liitetä avaimeen, koska käännösmuokkain pystyy oikaisemaan niissä esiintyvät erot.

Esimerkiksi virkkeen *Ville kirjoittaa esitelmää työhuoneessaan* avaimeksi saadaan *kirjoittaa 1 2 26*, jossa numerot 1, 2 ja 26 vastaavat virkkeen lauseenjäsenten relaatioita.

Käytettäessä edellä kuvattua hakutermiä, joka jakaa tietokannan samajuuristen käännösyksikköiden ekvivalenssiluokkiin, syntyy helposti ongelmia tietokannan kasvaessa: tällöin myös ekvivalenssiluokkien koot kasvavat, koska tietokannasta löytyy useita samajuurisia puita. Esimerkiksi suomen *olla*-verbijuuriset virkkeet muodostavat käytännössä erittäin suuren joukon. Tällöin haku hidastuu, koska joudutaan tarkemmin tutkimaan suuremman käännösyksikköjoukon soveltuvuus. Tätä hidastumista voidaan optimoida määrittelemällä hakutermi automaattisesti tarkentuvaksi tai määrittelemällä usempia indeksejä.

Automaattisesti tarkentuvaa avainta varten voidaan luoda relaatio usein esiintyvistä hakutermeistä, jotka vaativat tarkennusta. Kun huomataan, että jokin ekvivalenssiluokka on liian suuri, lisätään sen avain tähän relaatioon ja sen lingvistiset hakutermit lasketaan uudelleen. Nyt avainten muodostamiseen käytetään myös esimerkiksi juuren välittömiä lapsia. Ratkaisu on dynaaminen, mutta avaimien uudelleen laskeminen on hidasta.

Yksinkertaisempi ratkaisu on käyttää kaksi- tai useampitasoisia indeksejä. Haku tehdään ensin tarkimman indeksin avulla. Jos haku ei tuota tuloksia, etsitään toisen indeksin avulla, jossa avaimet on löyhemmin määritelty ja osumia tulee enemmän. Tarkin hakutermi muodostetaan puun kaikista relaatioista. Sen on tarkoitus palauttaa ne virkkeet, joista hyvin todennäköisesti saadaan viimeistään käännösmuokkaimella oikea käännös.

Työn lopussa olevassa testituloksia esittelevässä kappaleessa testataan sekä yksitasoisen että kaksitasoisen indeksoinnin käyttäytymistä suurella tietokannalla.

Puun muuntaminen vertailumuotoon

Ennen puiden vertailua ja tallettamista tietokantaan ne muutetaan taulukkoesitysmuotoon, jossa solmun lasten järjestys on normalisoitu. Solmut järjestetään niiden relaation eli lauseenjäsentiedon ja ennalta määrätyn relaatioiden preferenssilistan mukaan. Tässä preferenssilistassa relaatiot on jaettu lisäksi ekvivalenssiluokkiin. Luokkien järjestyksellä ei ole kovin suurta merkitystä, riittää että järjestys on ennalta määrätty ja että juurirelaatio on ensimmäisenä. Samaan ekvivalenssiluokkaan kuuluvat sellaiset relaatiot, joiden painojen on arvioitu olevan yhtä suuret haun kannalta. Esimerkiksi genetiiviattribuutti ja adjektiiviattribuutti kuuluvat samaan luokkaan. Ekvivalenssiluokkia käytetään varsinaisessa puiden etäisyyslaskennassa.

esim. relaatioiden preferenssilistan alusta:

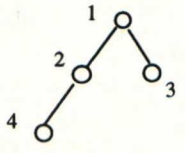
Juuri	= 1 = ko. relaatioluokan tunniste
Subjekti	= 2
Objekti	= 3

Adjektiiviattribuutti = 4
Genetiiviattribuutti = 4
...
Tämän järjestyksen perusteella virkkeen *Pekka kirjoittaa kirjaa* subjekti *Pekka* tulisi taulukossa ennen objektia *kirjaa.*

Vertailumuodossa oleva puu koostuu kolmesta taulukosta: ensimmäisessä taulukossa (**parentid**) esitetään solmun i =1...N isä:

esim.

tree->parentid [i=1] = 0 = juuri
tree->parentid [i=2] = 1
tree->parentid [i=3] = 1
tree->parentid [i=4] = 2



Taulukon ensimmäinen solmu on aina puun juuri. Jos kyseessä on metsä, jossa on m puuta, ovat solmut i=1..m juuria. Solmut ovat puussa rintamahakujärjestyksessä (Breadth First). Niinpä käymällä taulukko indeksien mukaisessa järjestyksessä tehdään puun leveyshaku. Jos jollain solmulla j on useita lapsia (i_1, i_2,..., i_N ; tree->parentid[i_1]= ...=tree->parentid[i+1] = j), jotka kuuluvat samaan relaatioiden ekvivalenssiluokkaan, ovat nämä solmut taulukossa peräkkäin ($i_1 = i$, $i_2 = i+1$,..., $i_N = i+N$).

Toisessa taulukossa (**node_attr**) ovat solmujen attribuutit. Yhden solmun attribuutit esitetään merkkijonotaulukkona.

Kolmas taulukko (**leftchild**) luodaan juuri ennen vertailua. Sillä esitetään solmun vasemmanpuoleisin lapsi. Tämä taulukko mahdollistaa puun läpikäymisen ylhäältä alaspäin. Leftchild-taulukkoa ei talleteta tietokantaan, koska se voidaan muodostaa parentid-taulukosta lineaarisessa ajassa.

Jäsennyspuiden etäisyyden laskeminen

Jäsennyspuiden vertailun tavoitteena on laskea kahden virkkeen tai virkkeen osan etäisyyden lukuarvo, joka kuvaa niiden merkitysten eroa ja siten myös käännösten eroa. Mitä pienempi etäisyys on, sitä enemmän tekstit ja niiden käännökset muistuttavat toisiaan. Jäsennyspuiden etäisyyksiä tarvitaan haun toisessa vaiheessa (kuva 5.5).

Älykkäässä käännösmuistissa riittää, että selvitetään puiden **suhteellinen** etäisyys käännettävänä olevaan virkkeeseen. Absoluuttinen etäisyysmittari olisikin hyvin vaikea määritellä luonnollisen kielen virkkeille dimensioiden suuren määrän takia.

Jäsennyspuiden etäisyyksiä joudutaan potentiaalisesti laskemaan usein, koska käännettäessä joudutaan virkkeille tai virkkeiden osille tekemään useita vertailuja. Kahden solmun attribuuttien vertaaminen on kohtalaisen työlästä, koska attribuutteja on useita ja niiden arvot voivat olla pitkiäkin merkkijonoja. Niinpä algoritmin aikavaativuus ilmaisee melko suoraan sen toimintanopeuden käytännössä. Vertailun pitää siis olla suhteellisen nopea ja toisaalta sen pitää pystyä erottamaan kielellisesti perustellut yhtäläisyydet ja erot. Ratkaisusta tulee välttämättä kompromissi näiden kahden tekijän välillä, koska puiden etäisyyksien laskeminen on yleisesti raskas ongelma ja toisaalta kielen ilmausten rikkaus on suuri.

Kahden puun etäisyys on määritelty edellä niiden minimikuvauksessa toisiaan vastaavien solmujen välisten etäisyyksien painotetuksi summaksi. Etäisyyden laskemiseksi algoritmin tulee löytää tämä puiden välinen minimikuvaus.

Algoritmi saa syötteenä vertailtavat puut vertailuun soveltuvassa taulukkoesitysmuodossa ja relaatioiden ekvivalenssiluokat (alla relaatioluokat).

```
Algoritmi          : Normalisoitujen jäsennyspuiden etäisyyden mittaus
Input              : Puu1 ja puu2 normalisoidussa BF-järjestetyssä
                     taulukkoesitysmuodossa, relaatioluokat, attribuuttien painot
Output             : etäisyys Puu1-Puu2 ja kuvaus Puu1->Puu2
-------------------------------------------------------------------------

Begin
 etäisyys = 0
 i1=0                    /* = Puu1:n juuri */
 Alusta kuvaus: aseta ensimmäiset juuret kuvautumaan toisikseen;
 while ( i1 < solmujen määrä puu1:ssä)
   {
        while (i1 on merkattu tuhotuksi)
          i1++

        lc1 = viimeinen i1:n veli
        level = i1:n syvyys puussa Puu1
        i2 = i1:n isää kuvauksessa vastaavan juuren 1:n lapsi.
        if (i2:ta ei löydy tai se on tuhottu)
          lisää kuvaukseen ja etäisyyteen solmujen i1...lc1 ja niiden lasten
          tuhoaminen
        else
         {
          lc2 = viimeinen i2:n veli

          while (i1<=lc1 or i2<=lc2)
            {
             srid1 = i1:n relaation relaatioluokan tunniste
             lc11 = i1:n viimeinen veli, joka kuuluu srid1:een

             srid2 = i2:n relaation relaatioluokan tunniste
             lc22 = i2:n viimeinen veli, joka kuuluu srid2:een

             if ( srid1 == srid2 )
               {
                Etsi minimipermutaatio se. solmujen  i1...lc11 ja i2..lc22
                välisen kuvauksen etäisyys on minimissä. Lisää permutaatio
                kuvaukseen ja etäisyyteen
                i1 = lc11+1
                i2 = lc22+1
               }
              else
               {
                if (srid1 < srid2)
                 {
                  lisää kuvaukseen ja etäisyyteen solmujen i1...lc11 ja sen
                  lasten tuhoaminen (alipuineen)
                  i1 = lc11+1
                 }
                else
                 {
```

```
                    lisää kuvaukseen ja etäisyyteen solmujen i2..lc22 sen lasten
                    tuhoaminen (alipuineen)
                    i2 = lc22+1
                  }
              }
            }
        }
    }

  lisää kuvaukseen ja etäisyyteen puu2:n vierailemattomien solmujen
  tuhoaminen.

  suhteuta etäisyys virkkeiden pituuteen
End
```

Algoritmi etenee puuta Puu1 leveyshakujärjestyksessä. Se luo kuvaustaulukon map1. Kun i=1..N on Puu1:n solmu, map1[i] kuvaa sitä Puu2:n solmua, joksi i kuvautuu. Map1[i] = 0, jos solmu i tulee poistaa. Puun Puu2 tuhottavat solmut esitetään taulukossa map2.

Jokaisen läpikäydyn tason jälkeen sen solmujen kuvautuminen Puu2:n solmuihin on valittu. Solmut voivat kuvautua vastaavalla syvyydellä puussa Puu2 oleviin solmuihin. Olkoon isä(k) solmun k isä. Jos kuvaukseen lisätään map1[i]=j, pitää joko i:n ja j:n olla juuria tai map1[isä(i)]= isä(j).

Alipuiden minimipermutaatio valitaan heuristisesti vertailemalla niiden juurien etäisyyksiä. Muiden solmujen etäisyyksiä ei käytetä valintaan. Käytännössä permutointia tarvitaan pääasiassa etsittäessä attribuuttien tai adverbiaalien oikeaa järjestystä. Niillä on yleensä vähän alisolmuja. Ilman permutointia algoritmi olisi lineaarinen. Permutointi on hidasta, jos solmulla on paljon jälkeläisiä tietyssä relaatioluokassa. Niinpä algoritmiin on valittu tämä nopea heuristiikka.

Algoritmi käyttää puiden solmujen etäisyyden laskemiseen niiden attribuuttien tärkeyden painoa, joka annetaan erikseen. Se lisätään etäisyyteen, jos ko. attribuuttien arvot eroavat. Jotain attribuutteja varten joudutaan tekemään erikoisjärjestelyjä.

Attribuutin lajin ja arvon lisäksi solmujen etäisyyteen vaikuttaa niiden paikka puussa: mitä korkeammalla solmu on puussa, sitä suurempi paino. Tämä on perusteltua, sillä puun yläosaan sijoittuvat lauseenjäsenet ja puun alaosaan lauseenjäsenten attribuutit, joiden vaikutus puiden väliseen semanttiseen etäisyyteen on vähäinen.

Etäisyys kasvaa eksponentiaalisesti erojen lukumäärän mukaan ennalta määritetyn kertoimen avulla. Etäisyys suhteutetaan myös virkkeen pituuteen sanojen lukumäärään perustuvan kertoimen avulla. Algoritmi palauttaa etäisyyden 0, jos virkkeet ovat samanlaisia tai jos vain niiden muotoilu eroaa. Etäisyys on suurempi kuin 0, jos virkkeet eroavat jollain tavalla.

Käännösyksiköiden pintamuotojen sumea vertailu

Myös käännösyksiköiden pintamuotojen sumeaa vertailua tarvitaan älykkään käännösmuistin tuotantoversiossa, koska kaikkien kielien depedenssijäsentimiä ei ole tarjolla. Toisaalta käännösmuistia voidaan haluta käyttää myös muiden kielien kanssa. Ainoastaan sanan käsitettä voidaan käyttää hyväksi vertailussa, muuten kyseessä on puhdas merkkijonovertailu. Vertailuun käytetään yleisesti tunnettuja merkkijonovertailualgoritmeja [Jokinen et al., 1991].

Käännösyksiköiden hallinta tietokannassa

Käännösmuistin prototyyppi on rakennettu **Berkeley DB**-tietokantaa käyttäen [Berkeley DB-specification, 1993]. Prototyypin perusteella tietokannan suorituskyvylle asetettavat tavoitteet selviävät paremmin. Samalla voidaan testata menetelmän soveltuvuutta.

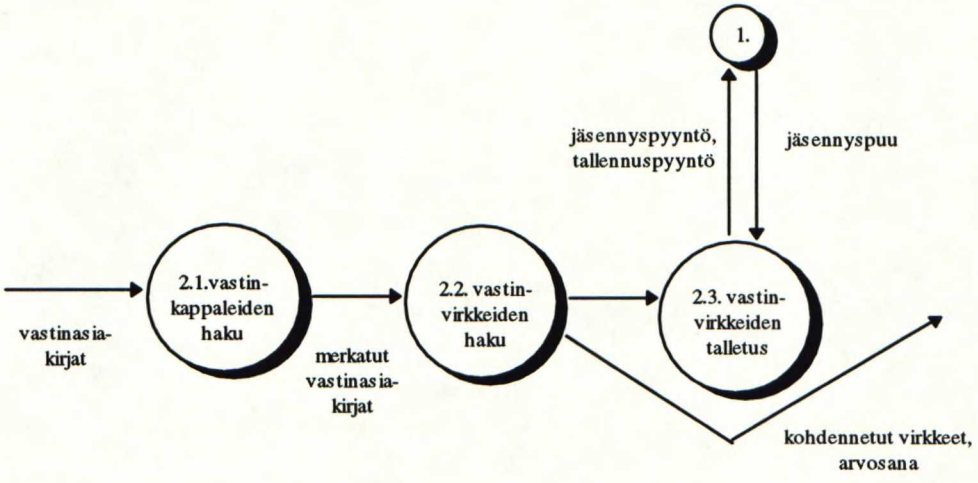
Tietokanta koostuu kahdesta pääosasta: tietovarastosta, jossa osoitukseen käytetään Berkley DB:n recno-osoitusmenetelmää. Siinä jokainen tietokantaan talletettava tietue saa järjestysnumeronsa mukaan kokonaisluvun avaimekseen. Tietokannan toinen osa koostuu indekseistä, joita on kaksi. Ne ovat hajautustaulukoita, joissa älykäs hakutermi kuvataan joukkoon tietovaraston recno-tunnisteita.

Lisättäessä uutta käännösyksikköä haetaan ensin indekseistä virkkeen hakutermiä vastaavat recno-tunnistejoukot. Näihin joukkoihin liitetään tietovarastoon lisätyn käännösyksikön saama recno-tunniste. Joukot talletetaan takaisin indeksitaulukoihin.

Kun tietokantaan tehdään haku, haetaan ensin indekseistä recno-tunnistejoukko. Joukon tunnisteita vastaavat käännösyksiköt haetaan tietovarastosta ja lisätään listaan, joka palautetaan.

Käännösmuistin tuotantoversiossa on lisäksi käänteisindeksi, joka mahdollistaa haun virkkeiden kaikkien sanojen mukaan. Käänteisindeksi kuvaa yksittäisen sanat recno-tunnistejoukkoon, jossa on lueteltu ne virkkeet, joissa sana esiintyy.

5.2. Vastinvirkkeiden kohdentaminen



Kuva 5.8. (DFD taso 2) Vastinvirkkeiden kohdennusjärjestelmän tietovuokaavio

Vastinvirkkeiden kohdentamista tarvitaan syötettäessä käännösmuistiin ihmiskääntäjän tekemiä käännöksiä. Jos käännösmuistia käytetään TranSmartin kääntämien ja ihmisen tarkastamien virkkeiden tallettamiseen, on kohdentaminen jo tehty eikä järjestelmää tarvita.

Vastinvirkkeiden kohdentamisjärjestelmä etsii suomen- ja englanninkieliset vastinvirkkeet syötteenä saamistaan asiakirjoista. Englanninkielisen asiakirjan pitää olla suomenkielisen käännös. Algoritmi antaa lisäksi jokaiselle löytyneelle vastinvirkeparille arvosanan. Arvosana kuvaa sitä, kuinka hyvänä algoritmi pitää löytynyttä kohdennusta.

5.2.1. Luonnollisen kielen virkkeiden kohdentamisen menetelmiä

Tunnetuin luonnollisen kielen virkkeiden kohdentamisen menetelmä on Galen ja Churchin (1993) kehittämä virkkeiden merkkimääriin ja dynaamiseen ohjelmointiin perustuva algoritmi. He ovat julkaisseet C-kielisen ohjelmansa. Algoritmi toimii seuraavalla tavalla :

1. Teksteihin merkitään hard-delimiter -merkit, joilla tekstit jaetaan kappaleisiin. Kappaleille pitää löytyä 1-1 vastaavuudet.
2. Tekstiin merkataan etukäteen myös virkkeiden rajat soft-delimiter -merkeillä.
3. Algoritmi etsii vastinvirkkeet tai vastinvirkeryhmät virkkeiden pituuksiin perustuen. Se hakee kappaleen sisällä sellaisen kohdennusryhmän, jonka kokonaiskustannus on pienin. Yhden kohdennuksen kustannus määritellään sen mukaan, kuinka hyvin sen suomen- ja englanninkielisten virkkeiden pituus vastaa ennalta määrättyä merkkimääräsuhdetta.

Algoritmi ei löydä 1:0 tai 0:1 vastaavuuksia - siis sellaisia virkkeitä, joita ei ole toisessa tekstissä lainkaan. Myöskään kaikki 2:1 tai 1:2 vastaavuudet eivät löydy. Galen ja Churhin algoritmi on helppo virittää uudelle kieliparille - riittää, että merkkimäärien suhde määritellään uudestaan.

Yleisemminkin kohdennusalgoritmit perustuvat siihen, että ensin etsitään suuremmat vastinyksiköt (esim. kappaleet) ja sitten siirrytään pienempiin, jopa aina sanatasolle saakka. Sen lisäksi, että voidaan käyttää virkkeiden pituuksia vastinvirkkeiden löytämiseen, voidaan käyttää myös ns. ankkurisanoja (*anchor words* tai *cognates*), jotka kääntyvät samalla tavalla eri teksteissä. Ankkurisanojen avulla etsitään vastinvirkkeet siten, että näistä pitää löytyä vastinankkurisanat. Näitä sanoja ovat mm. erisnimet, koodit, luvut, lyhenteet, päiväykset, lainasanat, monet partikkelit (ja, tai..) sekä välimerkit. Myös muiden sanojen käyttö on mahdollista, jos niille voidaan määritellä selkeästi rajattu käännösvastinejoukko. [Simard, 1993].

Eräät järjestelmät käyttävät myös termien tai sanaluokkien esiintymistiheyttä vastinvirkkeiden kohdentamiseen [Ahrenberg, 1995].

Martin ja Röscheinsen (1993) perustavat kohdentamisen tilastolliseen menetelmään ja tekstin useampaan läpikäyntiin. Joka kierroksella kerätään sekä vastinvirkkeitä että vastinvirkkeissä esiintyviä vastinsanapareja. Usein vastinvirkkeissä esiintyvät sanaparit oletetaan toistensa käännösvastineiksi. Seuraavalla kierroksella voidaan virkkeiden tasausta tarkentaa käyttämällä edellisellä kierroksella tarkentunutta sanastoa.

Käytännössä ihmisten tekemiä käännöksiä on vaikea kohdentaa, koska vastinvirkkeet eivät seuraa toisiaan riittävän selvästi. Virkkeet voivat olla uudelleenjärjestettyjä tai osa niistä voi puuttua kokonaan toisesta tekstistä. Davis, Dunning ja Odgen [1994] ovat tutkineet kohdentamista käytännönläheisillä englanti-espanja korpuksilla. He ovat päätyneet tarkentamaan kohdentamista sanastojen, numeroiden, merkkijonovertailun ja N-grammien vertailun avulla. Heidän algoritminsa ei kuitenkaan sellaisenaan toimi suomi-englanti teksteille ja vaatisi kieliparikohtaista virittämistä mm. sanastojen osalta.

5.2.2. Kohdentamisjärjestelmän toteutus

5.2.2.1. Manuaalinen kohdentaminen

Vastinkäännösyksiköiden syöttämiseen käännösmuistiin on kaksi tapaa: A. käsinsyöttö ja B. automaattinen syöttö. Käsisyötössä käyttäjä määrittelee etukäteen, mitkä käännösyksiköt hän haluaa talletettavan käännösmuistiin ja antaa tai valitsee niiden käännökset. Järjestelmä vain tallentaa käyttäjän antamat tiedot.

5.2.2.2. *Automaattinen kohdentaminen*

Virkkeiden tunnistaminen

Virkkeiden tunnistamiseen käytetään TranSmartin valmista virkkeentunnistinta. Sitä voidaan käyttää myös englannin virkkeiden tunnistamiseen. Ongelmaksi jäävät tällöin pisteelliset lyhenteet. Jos lyhenne loppuu pisteeseen ja seuraava sana alkaa isolla kirjaimella, ei voida aina olla varmoja onko kyseessä virkeraja vai esim. lyhenne ja erisnimi (esim. *Pituus on 121 mm. Holopainen mittaa sen / Siellä oli mm. Holopainen*). Tällaisia pisteellisiä lyhenteitä, joita seuraa isolla alkukirjaimella alkava sana esiintyy kuitenkin niin harvoin, ettei niitä tässä työssä oteta huomioon. Tämän lisäksi kohdennusalgoritmi kykenee yhdistämään väärin katkaistuja virkkeitä. Jatkossa virkkeentunnistimeen on mahdollista liittää listoja erilaisista pisteellisistä lyhenteistä.

Vastinvirkkeiden kohdentaminen

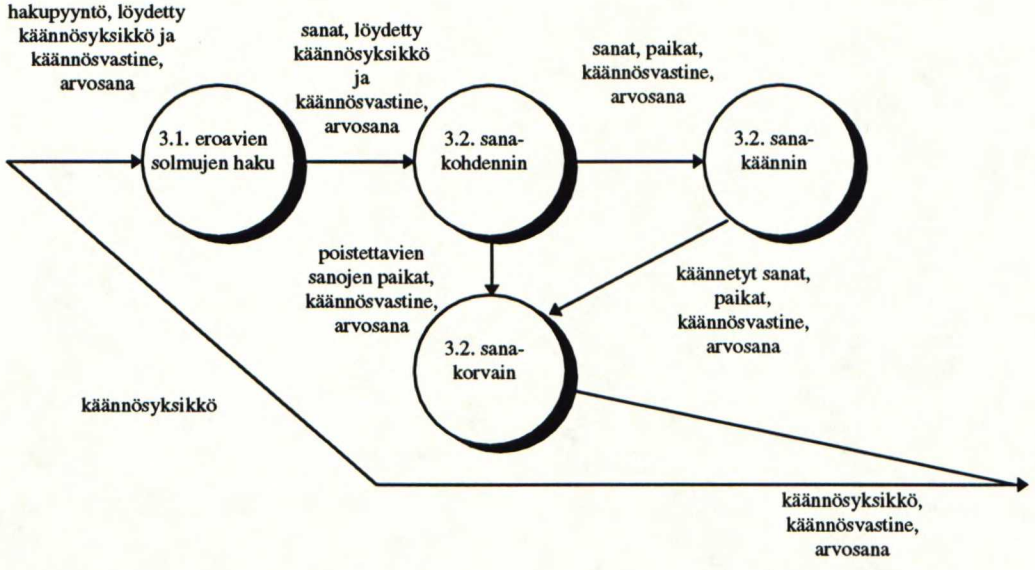
Vastinvirkkeiden kohdentaminen tehdään Galen ja Churchin menetelmällä. Menetelmää korjataan siten, että SGML-merkintöjä ei lasketa virkkeiden merkkimäärään. Galen ja Churchin menetelmä on varsin kieliriippumaton ja soveltuu suhteellisen hyvin suomi-englanti -kohdentamiseen.

Järjestelmä antaa jokaisesta kohdennuksesta arvosanan, joka perustuu siihen, kuinka hyvin saatu kohdennus vastaa ennalta määrättyä merkkisuhdetta. Arvosana on sitä suurempi, mitä huonompi kohdennus on.

Vastinvirkkeiden kohdennusjärjestelmä tuottaa tiedoston, joka vastaa muodoltaan käsintehtyä kohdistustiedostoa. Se voidaan syöttää käännösmuistiin.

5.3. Käännösmuokkain

5.3.1. Käännösmuokkaimen rakenne



Kuva 5.9. (DFD taso 2) Löydettyjen käännösvastineiden korjaaminen lopulliseen muotoonsa

Käännösmuokkaimen tehtävä on korjata osittain oikeasta esimerkkikäännöksestä lopullinen kohdekielinen virke, joka vastaa käännettävää virkettä. Muokkaimen toiminta perustuu siihen, että jos käännettävä virke ja tietokannasta löytyvä virke ovat rakenteellisesti riittävän lähellä toisiaan, on todennäköistä, että pienillä sanastokorjauksilla esimerkkikäännöksestä saadaan halutun virkkeen käännös.

5.3.2. Suomi-englanti käännösmuokkaimen periaate

Käännösmuokkaimen periaatteet noudattavat lähteessä Arnola (1995b) esitettyjä ajatuksia. Käännösmuokkaimen toiminta voidaan nähdä käännettävän virkkeen ja esimerkkikäännöksen etäisyyden pienentämisenä. Tämä voidaan jakaa kolmeen tapaukseen: 1. leksikaalisen etäisyyden poistaminen. 2. sanan attribuuttieron poistaminen ja 3. relaatioetäisyyden poistaminen. Viimeksi mainittua käsitellään vain attribuuttien ja adverbiaalien tiettyjen relaatioerojen poistamisessa.

Leksikaalisen etäisyyden poistaminen

Jos käännettävän virkkeen ja käännösesimerkin suomenkielinen jäsennyspuu ovat topologisesti samanlaisia ja vain solmujen lekseemit eroavat, voidaan käyttää seuraavaa algoritmia käännösesimerkin englanninkielisen vastineen muokkaamiseen:

1. Kullekin solmulle, joka on erilainen käännettävän virkkeen ja esimerkkikäännöksen suomenkielisen virkkeen jäsennyspuissa, haetaan kohdekielinen käännösvastine.
2. Esimerkkikäännöksen englanninkielisestä virkkeestä paikallistetaan ko. solmua vastaava sana
3. Sana korvataan lekseemin käännöksellä.

Oletetaan esimerkiksi, että käännösmuistiin on talletettu *Pekka istuu penkillä* ja sen käännös → *Pekka is sitting on a bench.* Pitää kääntää virke *Ville istuu penkillä.* Nyt virkkeissä poikkeavat vain subjektien lekseemit toisistaan: *Pekka* ja *Ville.* Käännökseksi saadaan paikantamisen ja korvaamisen jälkeen *Ville is sitting on a bench.*

Sanan attribuuttieron poistaminen

Määritellään ensin attribuuttiero (AD) nollasta poikkeavaksi ja pienemmäksi kuin ∞, jos solmujen **paikallissijaisten substantiivien** sijat poikkeavat tai jos **verbien aikamuoto tai tapaluokka** poikkeavat käännettävässä virkkeessä ja esimerkkikäännöksessä. Attribuuttiero voidaan poistaa, jos se on pienempi kuin ∞:

1. Kullekin **substantiivisolmulle**, jolle AD>0, suomenkielisen solmun **oletuspreposition** (alla) paikallistetaan englanninkielisestä virkkeestä.
2. Paikallistettu prepositio korvataan käännettävän noodin sijaa vastaavalla **oletuspreposition**.

Oletuspreposition:

inessiivi : in, into; elatiivi : from ; illatiivi: into, onto
adessiivi : on, onto, in, into: ablatiivi: from, allatiivi: into, onto
essiivi : as ; translatiivi: for : abessiivi: without

Esim. käännettävänä on *Pekka tuli talosta* ja tiedossa on *Pekka meni taloon* → *Pekka went into the house.* Poistamalla *talo*-solmujen attribuuttiero sekä *tulla*- ja *mennä*-solmujen leksikaalinen etäisyys saadaan *Pekka came from the house.*

Verbien attribuuttieron poistamista varten oletetaan, että sanastoon on talletettu aikamuodoltaan tai persoonaltaan poikkevat verbimuodot, ja että muut verbimuodot saadaan produktiivisten sääntöjen avulla.

1. Kullekin **verbisolmulle**, jonka AD>0 ja poikkeama on **aikamuodossa** tai **persoonassa**, haetaan sanastosta esimerkkikäännöksen suomenkielisen solmun käännösvastine. Sen avulla paikallistetaan sana englanninkielisessä virkkeessä. Paikallistettu sana korvataan vastaavalla tavalla saadulla käännettävän solmun käännöksellä.
2. Kullekin **verbisolmulle**, jonka AD>0 ja poikkeama on **tapaluokassa** haetaan sanastosta ja produktiivisten sääntöjen mukaan esimerkkilauseen solmua vastaava sanaliitto. Sanaliitto paikallistetaan ja korvataan vastaavalla tavalla saadulla käännettävän virkkeen solmua vastaavalla sanaliitolla.

Esim. käännettävänä on *Pekka voi kävellä kotiin* ja tiedossa on *Pekka kävelisi kotiin→ Pekka would walk home*. Voida-verbi on normalisoitu kävellä verbin attribuutiksi Oksaimessa, joten suomenkielisten virkkeiden jäsennyspuut ovat topologisesti samanlaisia. *Kävelisi*-solmua vastaava sanaliitto *would walk* paikannetaan ja korvataan *voi kävellä*-solmun käännöksellä ja saadaan *Pekka can walk home*.

Adverbisten adverbiaalien ja adjektiiviattribuuttien relaatioeron poistaminen

Jos adjektiiviattribuuttien määrät poikkeavat, voidaan virkkeiden etäisyys poistaa hallitusti, koska sekä suomen- että englanninkielessä adjektiiviattribuutit sijaitsevat pääsanansa edessä. Tästä syystä niitä voidaan produktiivisesti lisätä tai poistaa englanninkielisestä virkkeestä käännettävänä olevan virkkeen mukaan.

1. Jos käännettävässä virkkeessä jollain sanalla on enemmän adjektiiviattribuutteja kuin vastaavalla solmulla esimerkkikäännöksessä, paikallistetaan esimerkkikäännöksen adjektiiviattribuuttien pääsana englanninkielisestä virkkeestä. Paikallistetun sanan eteen lisätään käännettävän virkkeen lisäattribuutit.
2. Jos käännettävässä virkkeessä jollain sanalla on vähemmän adjektiiviattribuutteja kuin vastaavalla solmulla esimerkkikäännöksessä, paikallistetaan esimerkkikäännöksen ylimääräiset adjektiiviattribuutit englanninkielisestä virkkeestä ja poistetaan ne.

Esim. pitää kääntää *Iloinen mummo syötti lintuja*, kun tiedossa on *Mummo syötti pieniä lintuja → The granmother was feeding little birds*. Ensin huomataan lisäsolmu *iloinen*. Sen korjaamiseksi käännöksestä paikannetaan *grandmother* ja lisätään sen eteen *happy*. Sana *little* poistetaan ja saadaan käännös *The happy granmother was feeding birds*.

Myös adverbiaaliset adverbit voidaan produktiivisesti lisätä englanninkielisen virkkeen muiden adverbien joukkoon. Tämä tehdään kuten adjektiiviattribuuttienkin eron poistaminen: paikallistamalla sanat englanninkielisestä virkkeestä ja joko lisäämällä uudet vastineet verbin ja muiden adverbien perään tai poistamalla tarpeettomat sanat.

Jos älykästä käännösmuistia käytetään kääntäjän ainoana apuvälineenä interaktiivisesti, voidaan muokkaimessa olla hieman vapaamielisempiä ja sallia kaikkien muunlaistenkin solmujen vastinsanojen poistaminen. Samoin voidaan myös yrittää lisätä englanninkieliseen virkkeeseen kaikki käännösesimerkistä puuttuvien sanojen vastinsanat. Nämä muokkaukset eivät kuitenkaan aina ole kielellisesti oikeita, joten niitä ei voida tehdä, kun kääntäminen tapahtuu eräajona. Tällainen vapaampi muokkaaminen voi olla käyttökelpoinen menetelmä esimerkiksi käännettäessä suomesta muille kielille kuin englantiin.

Käännösmuokkaimella varustettua käännösmuistia voidaan pitää jonkinasteisena käännösjärjestelmänä, joka oppii rakenteellisia käännössääntöjä valmiista korpuksista. Esitetyillä periaatteilla ei päästä aina oikeaan käännöstulokseen. Ainakin interaktiivisesti käytettävässä käännösmuistissa on kuitenkin hyötyä jo siitä, että päästään lähemmäs oikeaa käännöstulosta.

5.3.3. Suomi-englanti käännösmuokkaimen toteutus

Käännösmuokkain toimii edellä esitettyjen periaatteiden mukaan. Ensin lasketaan puiden etäisyys. Jos sen lukuarvo on riittävän pieni, päätetään, että muokkaimen käyttö on mahdollista.

```
Algoritmi       : Suomi-englanti käännösmuokkain
Input           : Käännettävän virkkeen jäsennyspuu,
                  Esimerkkikäännöksen jäsennyspuu ja englanninkielinen
                  pintavirke
Output          : Käännettävän virkkeen englanninkielinen käännös tai
                  FALSE (=muokkaaminen epäonnistui)

-------------------------------------------------------------------------
Begin
        epäonnistunut_korjaus = FALSE;

        while ( korjattavia solmuja )
          {
            switch (erotyyppi)
              {
                case lekseemiero:
                case attribuuttiero:
                  {
                    newnode = käännettävän virkkeen solmu
                    oldnode = newnode:n vastinsolmu käännösesimerkissä
                    p = oldnode:n vastinsana käännöksessä
                    if (kohdennus onnistui)
                     {
                      tr = newnode käännösvastine.
                      if (käännösvastine löytyi)
                       {
                        korvaa p tr:llä
                       }
                      else
                         epäonnistunut_korjaus = TRUE;
                      }
                    else
                         epäonnistunut_korjaus = TRUE;
                  }
                case adjektiiviattribuuttimääräero :
                case adverbiaalimääräero :
                  {
                    if (esimerkkikäännöksessä ylimääräinen solmu)
                     {
                      oldnode = newnode:n vastinsolmu käännösesimerkissä
                      p = oldnode:n vastinsana käännöksessä
                      if (kohdennus onnistui)
```

```
                    {
                     poista p
                    }
                   else
                     epäonnistunut_korjaus = TRUE;
                   }
                 else     /* käännettävässä virkkeessä ylimääräisiä solmuja */
                  {
                   newnode=käännettävän virkkeen solmu
                   oldnode=newnode:n isän vastinsolmu käännösesimerkissä
                   p = oldnode:n vastinsana käännöksessä
                   if (kohdennus onnistui)
                    {
                     tr = newnode käännösvastine.
                     if (käännösvastine löytyi)
                      {
                       lisää tr p:n eteen (adj.attr) tai p:n perään (adverbiaali)
                      }
                     else
                      epäonnistunut_korjaus = TRUE;
                     }
                   else
                      epäonnistunut_korjaus = TRUE;
                  }
                }
            }
        palauta vastinvirke ja tieto
        onnistumisesta ( =NOT epäonnistunut_korjaus )
End
```

Muokkain hakee ensin puista eroavan, ylimääräisen tai puuttuvan solmun. Se paikallistaa poikkeavan solmun tai tarvittaessa sen isäsolmun vastinsanan käännösesimerkistä. Paikallistamiseen käytetään kaksikielisiä sanastoja ja englanninkielen morfologista analysaattoria. Jos yksikäsitteinen vastinsana löytyy, se korvataan käännettävän virkkeen sanan käännöksellä. Parhaimpaan tulokseen päästään, kun sana käännetään konteksiriippuvien sääntöjen avulla.

Jos käännösmuokkain ei onnistu kokonaan korjaamaan ko. virkettä, se palauttaa tästä tiedon. Käännösmuokkain palauttaa tällöin myös osittain korjatun virkkeen, josta on toivottavasti apua käännöstyötä tekevälle ihmiselle.

5.4. TranSmart-rajapinta

5.4.1. Tehtävät

TranSmart käsittelee käännettäviä virkkeitä puina omassa sisäisessä esitysmuodossaan. Jos käännösmuistin virkkeitä tai niiden osia halutaan hyödyntää TranSmartissa, pitää löytyneet mallikäännökset siirtää TranSmartin puihin. Se on tämän osajärjestelmän tehtävä.

Osajärjestelmän vastuulla on myös parhaan käännöksen valitseminen: se päättää, käytetäänkö käännösmuistista löytyvää käännöstä vai sääntöpohjaisen järjestelmän tuottamaa käännöstä.

5.4.2. Toteutus

Älykkään käännösmuistin liittämistä varten tehdään tarvittavat muutokset TranSmartin ohjausjärjestelmään. Käännösmuistille tehdään **alustus** ja **lopetuskutsut** sekä jokaista virkettä varten kaksi kutsua: toinen **jäsennyksen jälkeen** ja toinen varsinaisen **prosessin loppuun.**

Virkkeen jäsentämisen jälkeen jäsennyspuu muutetaan hakumuotoon. Se syötetään käännösmuistille. Jos muistista löytyy täsmälleen haettu virke, sen koko käännös siirretään jäsennyspuun juurisolmun käännökseksi ja muut solmut poistetaan. Käännössääntöjä ei tarvita.

Jos tietokannasta ei löydy täsmälleen oikeaa käännöstä, laitetaan lähin talteen. Tämän jälkeen virke käännetään sääntöjen avulla, koska käännösmuokkain saattaa tarvita sääntökäännettyä puuta seuraavassa vaiheessa. Jos prosessin lopussa ajettava käännösmuokkain pystyy korjaamaan käännösesimerkin, siirretään TranSmart-puuhun korjattu käännös, muuten puu jätetään ennalleen.

6. Testituloksia

6.1. Ydinkäännösmuistin ja käännösmuokkaimen testit

6.1.1. Testi 1 - Kuormitettavuus ja virkkeiden palautus

Testin periaate

Tässä testissä on tarkoitus selvittää älykkään käännösmuistin toimivuus ja kuormitettavuus. Testissä tarkistetaan käännösmuistin kyky palauttaa alkuperäinen talletettu virke suuresta tietokannasta eli saanti (engl. recall). Testimateriaalina toimii 27607 satunnaisesti valittua Kielikone Oy:n asiakasprojekteissa ja testeissä olevaa virkettä ja niiden koneellisesti käännetyt vastinvirkkeet. Virkejoukko koostuu hyvin erilaisista virkkeistä, siihen kuuluu 'tavallisten' virkkeiden lisäksi mm. otsikoita, kaavoja ja taukoiden rivejä. Virkkeet on valmiiksi kohdennettu englanninkielisten vastineidensa kanssa.

Testin aluksi virkkeet syötetään tietokantaan. Indeksoinnin toimivuutta tutkitaan sekä yksitasoisella että kaksitasoisella indeksoinnilla.

Testiympäristönä toimii kaikissa testeissä IBM PowerPC RS6000/230 16 megatavun RAM-muistilla ja AIX-käyttöjärjestelmällä.

Testitulokset

	Syöttö	Haku (recall) 1 indeksi	Haku (recall) 2-tasoinen indeksi
Aineiston koko	27 607 virkettä	27 607 virkettä	27 607 virkettä
Kokonaishakuaika	4h 10 min	n. 4 vuorokautta	5 h 30 min
Virkeen keskim.haku	0,5 s/virke	12,4 s/virke	0,7 s/virke

Kuva. 6.1. Kuormitustestin tulokset.

Testiaineiston koko tekstimuodossa on n. 6.4 megatavua. Jäsennyspuineen yms. tietoineen virkkeistä syntyy noin 36 megatavun Berkley DB-tietokanta. Yhden indeksin koko on noin 5 megatavua.

Tietokannan luomiseen kuluu noin 4 tuntia ja 10 minuuttia. Pääosa ajasta kuluu 27607 virkkeen jäsentämiseen. Yhden virkkeen lisäämiseen kuluu suunnilleen 0.5 sekuntia. Syötettäessä tietoa tarvitaan jäsennyksen lisäksi vain älykkäät hakutermit. Niiden luominen on hyvin nopeaa. Koska tietokanta on nopea, toisen indeksitaulukkon luominen ei aiheuta näkyvää hidastumista.

Yksitasoinen indeksointi osoittautuu nopeasti liian tehottomaksi ratkaisuksi näin suuressa tietokannassa. Yksitasoinen indeksointi pitää tehdä melko laveaksi, jotta se mahdollistaa sumean haun. Tämän seurauksena jokaista virkettä kohden löytyy haun ensimmäisessä vaiheessa (kuva 5.5) useita, joskus jopa satoja lähellä olevia virkkeitä, jotka kaikki joudutaan hakemaan levyltä ja järjestämään. Yhden virkkeen hakeminen kestää keskimäärin 12.4 sekuntia. Koko aineiston hakeminen veisi n. neljä vuorokautta, mutta testi keskeytettiin ja kokonaisaika laskettiin yhden virkkeen keskimääräisen hakuajan perusteella.

Kun kaksitasoisen indeksin tarkempi indeksointi muodostetaan jäsennuspuiden kaikista relaatioista, samoja avaimia tulee vain joillekin lyhyille otsikoille ja tietyille olla-verbijuurisille

lauseille. Niinpä hakukin on nopea, koska puiden vertailuja ei tarvita yleensä kuin yksi jokaista haettavaa virkettä kohden.

Kaikkien virkkeiden hakeminen kesti tietokannasta kaksitasoisella indeksoinnilla yhteensä n. 5 tuntia 30 minuuttia. Yhden virkkeen hakuun kuluu keskimäärin 0.7 sekuntia. Tähän nopeuteen päästään käytännössä vain asiakirjojen uusia, muuttumattomia versioita käännettäessä.

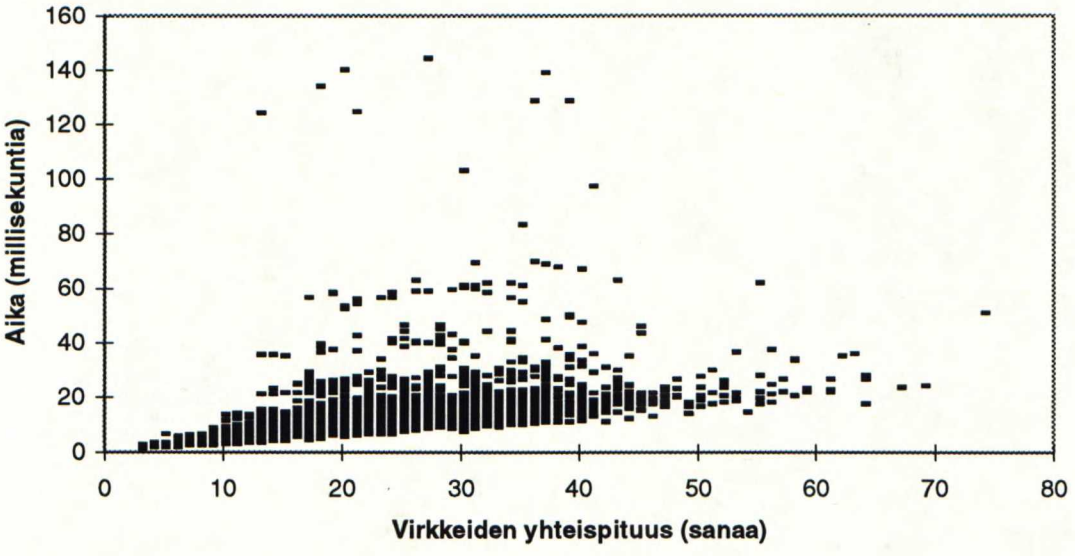
Testi paljasti jotain puutteita ja virheitä ohjelmakoodissa. Harvinaisimmat tulivat esiin vain muutamassa käännösyksikössä, vaikka aineiston koko on suuri. Virheiden korjaamisen jälkeen testi uusittiin ja kaikki 27607 tietokantaan talletettua virkettä löydettiin (recall = 100%, noise=0%).

6.1.2. Testi 2 - Puuvertailun aikavaativuus

Testin periaate

Tässä testissä on tarkoitus selvittää puuvertailun osajärjestelmän käytännön aikavaativuus. Yleinen puiden vertailu on todistettu kirjallisuudessa aikavaativuudeltaan eksponentiaaliseksi. Tässä työssä esitetyn algoritmin tulisi kuitenkin toimia lähes lineaarisesti. Solmujen permutointi hidastaa algoritmia. Testin tuloksesta nähdään likimäärin kuinka paljon permutointia käytännön virkkeitä vertailtaessa joudutaan tekemään.

Testitulokset



Kuva 6.2. Puiden vertailunopeus suhteessa virkkeiden pituuksiin.

Kuvassa 6.2. nähdään n. 4000 satunnaisesti valitun virkeparin etäisyyden laskemiseen kuluva aika suhteessa virkkeiden yhteispituuksiin. Pääosa virkepareista käyttäytyy likimain samalla tavalla eli lineaarisesti suhteessa niiden pituuteen. Joidenkin virkkeiden jäsennyspuut sisältävät kuitenkin enemmän samoja lauseenjäseniä tietyn solmun lapsina. Tällöin algoritmi joutuu permutoimaan, johon kuluu aikaa. Tulokset ovat odotettuja ja valituksi tulleet esimerkkivirkkeet muodostanevat riittävän kattavan aineiston.

Testi osoittaa kuitenkin myös, että on mahdollista melko helposti luoda sellainen virkepari, jossa permutointiin kuluu hyvin paljon aikaa, jopa kymmeniä sekunteja. Tästä syystä algoritmiin on syytä lisätä permutointia rajoittava ehto.

6.1.3. Testi 3 - Menetelmän soveltuvuus, sumea vertailu ja käännösmuokkain

Testien periaate

Testin tarkoituksena on selvittää älykkään käännösmuistin soveltuvuus oikeiden tekstien kääntämiseen. Hyöty vanhojen asiakirjojen uusia versiota käännettäessä on ilmeinen niissä tekstiosissa, joissa ei ole muutoksia tapahtunut. Tässä testissä tutkitaan, miten asiakirjassa tapahtuneet muutokset vaikuttavat älykkään käännösmuistin käytettävyyteen ja kuinka paljon käännösmuistista on hyötyä aivan uusien asiakirjojen kääntämisessä. Testi kohdistuukin käännösmuistin sumean vertailun ja käännösmuokkaimen toiminnan evaluointiin.

Testimateriaalina toimii sama 27607 virkkeen joukko kuin ensimmäisessä testissä. Näistä valitaan 184 virkettä, joita muokataan satunnaisesti käsin siten, että virkkeen pintamuoto ja merkitys hieman muuttuvat. Virkkeisiin esimerkiksi lisätään määreitä tai niistä poistetaan sanoja. Muokkaaminen pitää tehdä käsin, jotta tuloksena saadut virkkeet noudattaisivat suomen kielioppia.

Muokatut virkkeet käännetään älykkäällä käännösmuistilla. Testi onnistuu, jos alkuperäinen, muokkaamaton virke tai sitä parempi virke tarjotaan parhaaksi käännöspohjaksi. Muokkaimen tulee lisäksi muuttaa alkuperäisen virkkeen käännös muokattua vastaavaksi.

Sitä, miten käännösmuisti toimii oikeiden asiakirjojen uusien versioiden kääntämisessä, ei voida tutkia, koska saatavilla ei ole sopivaa aineistoa. Joidenkin muokattujen virkkeiden oletetaan kuitenkin edustavan myös asiakirjojen uusissa versiossa tapahtuneita muutoksia, kuten nimien vaihtumista tai määrittelyjen tarkentamista.

Testitulokset

	Korjattu oikein	Korjattu väärin	Ei osaa korjata	Ei mitään käännöstä
Kpl	34	4	118	28
%	18	2	64	15

Kuva. 6.3. 184:n käsin muokatun virkkeen haun tulokset.

Ilman käännöstä jää 28 virkettä, koska niiden juuri tai sitä lähellä olevat solmut ovat muuttuneet niin paljon, etteivät virkkeet enää kuvaudu samaan hakutermiin kuin alkuperäiset. Virkkeiden etäisyys voidaan kuitenkin laskea näillekin virkkeille. Kaikkien muokattujen virkkeiden etäisyys alkuperäiseen virkkeeseen vaikuttaa intuitiivisesti oikealta. Lopuille 156:lle (34+4+118) ehdotetaankin käännöspohjaksi alkuperäisen virkkeen käännöstä.

Tämä testi osoittaa, että menetelmän heikoin linkki on indeksointi, koska se saattaa pudottaa hyviä käännösehdotuksia pois löydettyjen virkkeiden joukosta. Testi uusittiin siten, että toisen tason indeksointiin käytettiin vain puiden juurien synonyymisanastosta haettua perustulkintaa. Tulokset ovat kuvan 6.4 mukaiset. Ne ovat paremmat, mutta aikaa kului enemmän.

	Korjattu oikein	Korjattu väärin	Ei osaa korjata	Ei mitään käännöstä
Kpl	46	4	130	4
%	25	2	71	2

Kuva. 6.4. 184:n käsin muokatun virkkeen haun tulokset hitaalla indeksoinnilla

Tässä testissä älykkään käännösmuistin hyöty verrattuna perinteisiin käännösmuisteihin näkyy suoraan 18%:ssa (tai 25 %:ssa) virkkeistä, koska niistä saadaan automaattisesti oikea käännös. Perinteisissä käännösmuisteissa ihminen joutuisi muokkaamaan myös näitä virkkeitä. Järjestelmän käytettävyyden kannalta olisi tarkoituksenmukaista, jos voitaisiin aina oikein päättää, onko muokattu käännös oikea vai väärä. Kuvassa 6.4 on esitetty jotain käännösmuokkaimen oikein muokkaamia virkkeitä. Käännettävää virkettä lähintä vastaavaa esimerkkikäännöstä muokkaamalla on saatu *tulos*-kohdassa oleva virke.

käännettävä virke: Kehitystyössä käytetään uusimpia suosituksia ja standardeja.
löytyi: Kehitystyössä käytetään uusimpia standardeja ja suosituksia.
esimerkkikäännös: In the development work some of the newest standards and recommendations are used .
etäisyys: 1.5786
tulos: In the development work some of the newest recommendations and standards are used .

---

käännettävä virke: 2.1 Järjestelmän soveltuvuus
löytyi: 2.1 Järjestelmän yleinen soveltuvuus
esimerkkikäännös: 2.1 General suitability of the system
etäisyys: 7.0833
tulos: 2.1 suitability of the system

---

käännettävä virke: DX 300 -JÄRJESTELMÄ UUDESSA PUHELINVERKOSSA
löytyi: DX 200 -JÄRJESTELMÄ PUHELINVERKOSSA
esimerkkikäännös: DX 200 SYSTEM IN TELEPHONE NETWORK
etäisyys: 7.1400
tulos: DX 300 SYSTEM IN new TELEPHONE NETWORK

---

käännettävä virke: Uudet keskukset liitetään toisiinsa XXX-siirtojärjestelmillä ja standardimerkinantoa käyttäen.
löytyi: Uudet keskukset liitetään toisiinsa XXX-siirtojärjestelmillä ja sovittua standardimerkinantoa käyttäen.
esimerkkikäännös: The new centres will be connected to each other with the XXX transport systems and using agreed standard signalling .
etäisyys: 0.9659
tulos: The new centres will be connected to each other with the XXX transport systems and using standard signalling .

---

käännettävä virke: Kaikki merkinanto käsitellään siten, ettei johtokohtaisia sovituslaitteita tarvita lainkaan.
löytyi: Kaikki merkinanto käsitellään siten, ettei johtokohtaisia sovituslaitteita tarvita.
esimerkkikäännös: All signalling is dealt with so that cable-specific adaptation devices will not be needed .
etäisyys: 0.9659
tulos: All signalling is dealt with so that cable-specific adaptation devices will not be needed at all .

---

käännettävä virke: 5 suurinta teollisuusryhmää :
löytyi: Neljä suurinta teollisuusryhmää :
esimerkkikäännös: The four biggest industrial divisions :
etäisyys: 1d=1.0625
tulos: The 5 biggest industrial divisions :

Kuva. 6.5. Jotain oikein kääntyneitä esimerkkikäännöksiä.

Tässä testissä käännösmuokkain tekee virheen neljässä virkkeessä. Kyseessä on joitain huomioimatta jätettyjä poikkeustapauksia, kuten sanan attribuuttien järjestyksen aiheuttamat ongelmat. Erään käännettävän virkkeen loppu kuuluu:

... on korvattu uudella DX200 -keskuksella

ja lähin esimerkkikäännös on:

... on korvattu DX 200 -keskuksella → ...is replaced with DX 200 exchange

Koska muokkain lisää nykyisellään uudet adjektiiviattribuutit suoraan pääsanansa eteen, saadaan käännökseksi

→ *...is replaced with DX 200 new exchange*, eikä
→ *...is replaced with new DX 200 exchange.*

Käytetty menetelmä mahdollistaa useiden esiin tulleiden virheiden korjaamisen. Jotkin virheet aiheutuvat kohdekielen ominaisuuksista. Vaikka suomenkieliset virkkeet ovat rakenteellisesti samoja, saatetaan niiden englanninkielisiin käännöksiin tarvita suurempia muutoksia. Tyypillinen esimerkki on englannin epämääräisen artikkelin *a/an* valinta sen pääsanan muututtua. Tällaisia virheitä voidaan korjata, jos virkkeistä pystytään paikantamaan suurempia sanaryhmiä.

Testivirkkeet luotiin käsin muokkaamalla valmiita virkkeitä. On vaikea arvioida, kuinka puolueeton testi on. Se paljastaa joitain järjestelmän puutteita ja vahvuuksia. Todellinen käytettävyyden ja hyödyllisyyden evaluointi voidaan kuitenkin tehdä vasta, kun järjestelmä toimitetaan asiakkaan käyttöön.

6.2. Kohdennusjärjestelmän testi

6.2.1.1. *Testien periaate*

Kohdennusjärjestelmän kykyä löytää vastinvirkkeet syötteenä olevista asiakirjoista testatatan esimerkkiaineistolla, joka kohdennettaan ensin käsin. Tämän jälkeen sama materiaali pilkotaan virkkeisiin ja kohdennettaan koneellisesti. Koneen antamia tuloksia verrataan käsin tehtyihin.

Kohdennuksia verrattaan laskemalla erojen määrä ja selvittämällä joidenkin syyt. Kohdennusjärjestelmä tuottaa tulosteena tekstitiedoston, jossa vastinvirkkeet tai vastinvirkeryhmät ovat peräkkäisillä riveillä. Myös käsin tehty tiedosto on tällainen. Erot voidaankin laskea Unixin diff- ja wc-komentojen avulla.

Koneen tekemän kohdennuksen oikeellisuuden lisäksi testataan myös sen antaman arvosanan luotettavuus. Kohdennusjärjestelmä antaa jokaiselle virkeparille arvosanan, jonka lukuarvo vaihtelee käytännössä 0:n ja 1500:n välillä. Lukuarvo on sitä suurempi, mitä huonommaksi algoritmi arvioi kohdennuksen. Testissä selvitetään, voidaanko tätä arvosanaa käyttää tarkastettavien kohdennuksien valintaan.

Esimerkkiaineistona toimii opetusalaa käsittelevä asiakirja ja ammattikääntäjän siitä tekemä englanninkielinen käännös. Suomalaisessa tekstissä virkkeitä on 444 ja englantikielisessä 489 kpl. Testimateriaalin englanninkielinen käännös on tehty kohtuullisen tarkasti suomenkielisesta tekstistä ja sen rakenteesta. Käännöksessä ei juurikaan ole ylimääräisiä virkkeitä eikä siitä ole poistettu virkkeitä. Vastinvirkeparit seuraavat siis kohtuullisen selkeästi toisiaan. Materiaali soveltuukin melko hyvin koneelliseen kohdentamiseen. Järjestelmän toiminnasta saisi

paremman kuvan suuremman testimateriaalin avulla, mutta kohdennettujen suomi-englanti käännösten saatavuus on huono.

6.2.1.2. Testitulokset

Nopeus

Käsin tehtynä kohdentaminen vie n. **2 tuntia**. IBM PowerPC Unix -työasemassa aikaa kuluu n. **30 sek,** vaikka kappalerajoja (ns. hard-delimiter) ei merkitä tekstiin. Jos kappalerajat merkitään, aikaa kuluu vain muutama sekunti. Algoritmi käyttää kappalerajoja ongelman osiin jakamiseen. Jos niitä ei ole merkitty, se joutuu laskemaan koko asiakirjojen kaikkien mahdollisten kohdennuksien välisen minimietäisyyden. Kokonaisuutena algoritmi on nopea, koska jo virkkeiden tunnistamiseen kuluu enemmän aikaa (n. 1 min).

Kohdennuksen oikeellisuus

Oikein	Väärin
380 kpl	49 kpl
89 %	11 %

Kuva. 6.5. Kohdennustestin tulokset.

Kohdennuksia syntyi koneellisesti 429 kpl, kun käsin tehtynä niitä oli 435 kpl. Algoritmi erehtyi 49 kertaa, joka on 11 % kaikista kohdennuksista. Oikein kohdentuu siis lähes **90%** virkkeistä. Suuri osa näistäkin virheistä (n. 10 kpl) johtuu virkerajojen puutteellisuudesta tekstin tietyssä kappaleessa.

Virheet kasaantuvat otsikoihin, listoihin yms. lyhyisiin virkkeisiin. Varsinaisten oikeinmuodostettujen virkkeiden joukossa virheitä on varsin vähän. Menetelmän kieliriippumattomuus ja nopeus huomioiden tuloksia voidaan pitää hyvinä. Toisaalta monet virheet sijoittuvat sellaisiin paikkoihin, missä on numeroita tai tms. ilmiöitä, joissa ankkurisana-menetelmien avulla päästäisiin parempiin tuloksiin.

Arvosanojen luotettavuus

Väärin kohdentuneet virkkeet saavat seuraavat arvosanat:

0	0	2	5	10	10
20	30	35	45	61	66
73	83	90	104	105	145
147	173	216	230	230	234
242	251	257	266	279	280
292	309	338	343	419	425
441	460	477	570	582	650
680	680	808	849	929	979
1060					

Kuva. 6.6. Väärin kohdentuneiden virkeparien arvosanat.

Virheelliset kohdennukset ovat ryhmittyneet joukoiksi, koska yksi virheellinen kohdennus aiheuttaa aina toisen, joskus useammankin lisävirheen. Jokaisen virheellisen ryhmän jokin kohdennus sai testissa arvosanan, jonka lukuarvo oli yli 200.

Oikein kohdentuneiden virkkeiden arvosanat jakautuvat siten, että suurin osa saa arvosanan, joka on alle 200 ja 25 kohdennusta saa yli 200 pistettä:

Kohdennuksien määrä	Arvosanaväli
380	0-200
18	200-300
7	300-654

Kuva 6.7. Arvosanojen jakauma.

Tämän testin perusteella kohdennusjärjestelmää voidaan pitää varsin käyttökelpoisena. Sitä voidaan käyttää hyväksi esimerkiksi siten, että koneellisen kohdentamisen jälkeen tarkastetaan käsin ne virkkeet joiden lähettyvillä jokin kohdennus on saanut arvosanakseen yli 200 pistettä.

7. Yhteenveto

Tämän työn lähtökohtana olivat TranSmart-konekäännösjärjestelmän käytössä asiakkaiden luona esiin tulleet adaptiivisuusongelmat ja käytettävyyspuutteet. Tavoitteena oli niiden selvittäminen ja ratkaisumahdollisuuksien hakeminen. Työn konkreettinen tulos on älykäs käännösmuisti, joka - päinvastoin kuin perinteiset käännösmuistit - perustuu jäsennyspuiden vertailemiseen ja sisältää käännösmuokkaimen.

Kokonaisuutena käytännön konekääntämisessä näyttää nousevan esiin kahdenlaisia ongelmia: yhteensopivuus- ja käytettävyysongelmia työkalujen kanssa sekä luonnollisen kielen luonteesta aiheutuvia lingvistisiä ongelmia.

Työkalujen yhteensopivuus- ja käytettävyysongelmat voidaan usein ratkaista tietoteknisin keinoin - ovathan ne usein tietotekniikan aiheuttamiakin. Muun muassa sanojen 'avoimuus' ja 'siirrettävyys' tulisi kuulua myös konekäännösjärjestelmien kehittäjien sanavarastoon.

Itse luonnollisen kielen hallitsemiseen ja kääntämiseen ei näytä löytyvän kattavaa mallia, joka voitaisiin siirtää nykyisten tietokoneiden kielelle. Yhdeksi ratkaisuksi nousee kuitenkin oppiva tai helposti opetettava, adaptiivinen konekäännösjärjestelmä. Käytännössä adaptiivisuuden 'harha' näyttää vielä syntyvän siitä, että järjestelmän kehittäjä on löytänyt mahdollisimman joustavan tavan kuvata jonkin ilmiön. 'Oikeasti' älykkään ja oppivan järjestelmän vaatima taustatietomäärä ylittää vielä nykyisten järjestelmien kapasiteetin.

Käännösmuisti edustaa oppimista suuren tietomäärän avulla. Oppiminen on hidasta, mutta virheitä ei juuri tapahdu opittua tietoa hyödynnettäessä. Toteutetussa älykkäässä käännösmuistissa on lisäksi otettu huomioon joitain luonnollisen kielen mahdollistamia yleistyksiä, jotka parantavat saantia tietokannasta. Yleistyksiä käytetään hyväksi muun muassa normalisointivaiheessa ja käännösmuokkaimessa.

Älykäs käännösmuisti toimii testeissä määritelmiensä mukaan. Testit osoittavat kuitenkin myös, miten usein luonnollinen kieli tarjoaa yllättäviä poikkeustapauksia, jotka saattavat unohtua järjestelmän määrittely- ja toteutusvaiheessa. Kokeilujen perusteella näyttää siltä, että älykäs käännösmuisti on mahdollista virittää sellaiseksi, että sen tarjoama käännös on riittävällä varmuudella oikea. Epäilyttävät käännökset voidaan antaa käyttäjän tarkistettavaksi.

Toisaalta menetelmä tarjoaa mahdollisuuden löyhempään parametrisointiin ja vapaampaan käännösten muokkaamiseen. Se voitaisiin helposti virittää muille kohdekielille ja saada siten aikaan yksinkertainen käännösjärjestelmä. Käännösmuistin muut jatkokehitysmahdollisuudet voisivat painottua esimerkkikäännösten pilkkomiseen pienempiin osiin, mikä mahdollistaisi paremman saannin tietokannasta. Kohdentaminen voitaisiin viedä nykyiseltä virketasolta aina lause- tai fraasitasolle asti. Niinhän monet ihmiskääntäjätkin oppivat.

8. Lähteet

Arnola, H., (haastatteluja), Kielikone Oy, Helsinki, 1995a.

Arnola, H., Älykäs käännösmuisti, Kielikone Oy, Helsinki, 1995b.

Ahrenberg, L., (haastattalu), Prof. of Computational Linguistics in Linköping, 1995.

Berkeley DB specifications, University of California Berkeley, 1993.

Boitet, C., Toward Personal MT: General Design, Dialogue Structure, Potential Role of Speech, *Proc. of COLING-90*, vol 3, Helsinki, 1990.

Booth, P., *An Introduction to Human-Computer Interaction*, Exeter, 1989.

comp.ai.nal-lang, vastauksia kyselyyn Internet-uutisryhmässä, 1995.

Davis, M.W., Dunning, T. E. and Odgen, W. C., Text Alignment in Real World: Improving Alignments of Noisy Translations Using Common Lexical Features, String Matching Strategies and N-Gram Comparisons, Computing Research Laboratory New Mexico State University, 1994.

Anon., EAGLES, Evaluation of Natural Language Processing Systems, Draft 1st of July, 1994.

Forsyth, R. (Ed.), Machine Learning - Principles and techniques, Chapman and Hall Ltd, London, 1989.

Furuse, O. and Iida, H., An Example-Based Method for Transfer-Driven Machine Translation, *Proc. of TMI-92*, Hikaridai, Kyoto, 1992.

Gale, W.A. and Church, K.W., A Program for Aligning Sentences in Bilingual Corpora, Computational Linguistics March 1993, vol 19, number 1, 1993.

Giguet, E., Multilingual Sentence Categorization according to Language, Computation and Language E-print Archive, 1995.

Hutchins, W.J., *Machine Translation: past, present, future*, Ellis Horwood Limited, Great Britain, 1986.

Hutchins, W. J. and Somers, H. L., *An Introduction to Machine Translation*, Academic Press, Cambridge, 1992.

Hutchins, W. J., Latest Developments in Machine Translation Technology: Beginning a New Era in MT Research, *Proc. of MT Summit IV*, Kobe, 1993.

Jokinen, P., Tarhio, J. and Ukkonen, E., A Comparion of Approximate String Matching Algorithms, Dep. of Computer Science, report A-1991.7, University of Helsinki, 1991.

Juola, P., Self-Organizing Machine Translation: Example-Driven Induction of Transfer Function, report CU-CS-722-94, University of Colorado, 1994

Jäppinen, H., Hartonen, K., Kulikov L., Nykänen, A. and Ylä-Rotiala A., KIELIKONE Machine Translation Workstation, *Progress in Machine Translation,* Nirenburg, S. (Ed.), Washington, D.C., 1993.

Kay, M. and Röscheisen, M., Text-Translation Alignment, Computational Linguistics 3/1993, vol 19, number 1, 1992.

Knight K., Chander, Haines, M., Hazzivassiloglou, V., Hovy, E., Iida, M., Luk, S.K., Okumura, A., Whitney, R. and Yamada, K., Integrating Knowledge Bases and Statictics in MT, USC/Information Science Institute, 1994.

Knight, K. and Chander, I., Automated Postediting of Documents, USC / Information Science Institute, 1994.

Kulikov, L., Tietokoneavusteinen kielenkääntäminen tiedonsiirrossa ja -haussa, SITRA Kielikone-projekti, 1991.

Kulikov, L., *Diplomityö: Konekäännös Puumuunnoksin*, Helsinki, 1990.

McLean, I. J, Example-Based Machine Translation using Connectionist Matching, *Proc. of the 4th International Conference on Theoretical and Methodological Issues in Machine Translation*, Montreal, 1992.

Nagao, M., Machine Translation: What Have We To Do?, *Proc. of MT Summit IV*, Kobe, 1993.

Nagao, M., *Machine Translation: How Far Can It Go?*, Oxford University Press, New York, 1989.

Newton, J., Translation Manager for Windows, Language International 6/6, 1994.

Nirenburg, S., Domashnev, C., Grannes and D. J., Two Approaches to Matching in Example-Based Machine Translation, *Proc. of Fifth International Conference on Theoretical and Methodological Issues in Machine Translation*, Kyoto, 1993.

Nuutila, P., puhelinkeskustelu käännösmuistista, Nokia Telecommunications Oy, 1995.

Piperidis, S. and Carayannis, G., Translearn: Interactive Corpus-Based Translation Drafting Tool, *Proc. of Language Engineering Conventions,* Paris, 1994.

Pressman, R. S., *Software Engineering - a Practioner's Approach 3 ed,* McGraw-Hill Inc, USA, 1992.

Anon., RTF-spesicication ver. 1.3, Microsoft Corporation, 1994.

Sato, S. and Nagao, M., Toward Memory-based Translation, *Proc. of COLING-90,* Helsinki, 1990.

Saukkonen, P., Haipus, M., Niemikorpi A. and Sulkala H., *Suomen kielen taajuussanasto*, WSOY, Porvoo, 1979.

de Schaetzen, C., Text Conversion, Language International 6/4, 1994.

Shasha, D., Wang, J. T., Zhang, K. and Shih, F. Y., Exact and Approximate Algorithms for Unordeded Tree Matching, IEEE Transactions on Systems, Man and Cybernetics, Vol 24, No. 4, April 1994.

Simard, M., Foster, G. F. and Isabelle, Pierre, Using Cognates to Align Sentences in Bilingual Corpora, *Proc. of TMI-92*, Montreal, 1992.

Slocum, J. (Ed.), *Machine translation systems*, Cambridge University Press, Great Britain, 1987.

Stonebraker, M., Future Trends in Data Base Systems, Postgress documentation, Department of Electrical Engineering and Computer Sciences University of California at Berkeley, 1988.

Sutcliffe, R. F. E., Boersma P., Bon, A., Donker, T. et al.(14 kpl), Beyond Keywords: Accurate Retrieval from Full Text Documents, *Proc. of Second Language Engineering Conventions,* London, 1995.

Tai, K., The Tree-to-Tree Correction Problem, Journal of the Association for Computer Machinery, Vol 26, No 3, July, 1979.

Anon., Asiantuntija asialle, Tietokonelehti joulukuu, 1994

Watanabe, H., A Method for Extracting Translation Patterns from Translation Examples, *Proc. of TMI-93*, Kyoto, 1993.

Wolinski, F., Vichot, F. and Dillet, B., Automatic Processing of Proper Names in Texts, Informatique CDC Caisse des Dépót et Cousiguatious, France, 1995.

Yasuhara, H., An Example-Based Multilingual MT System in a Conceptual Language, *Proc. of MT Summit IV*, Kobe, 1993.

Zhang, K. and Shasha, D., Simple Fast Algorithms for The Editing Distance Between Trees and Related Problems, SIAM Journal on Computing, vol 18, No. 6 December, 1989.